ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

Rendering volumetrico di dati provenienti da RM e TAC in realtà virtuale

Relazione finale in

Computer Graphics

Relatore

Dott.ssa Damiana Lazzaro

Presentata da

Xiang Xiang Ma

Sessione III
Anno Accademico 2016-2017

PAROLE CHIAVE

Realtà virtuale

HTC Vive

Volume rendering

Ray-Casting

Blender

Unity

Introduzione

L'obiettivo di questa tesi è quello di approfondire le tematiche relative allo sviluppo di applicazioni di realtà virtuale, prendendo come caso di studio la visualizzazione volumetrica, di dati provenienti dalle scansioni TAC e RM. Il progetto è stato realizzato per il visore di realtà virtuale *HTC Vive,* il quale è dotato di un avanzato sistema di tracciamento, che permette di interagire con l'ambiente virtuale in maniera precisa ed intuitiva. A differenza delle classiche periferiche come monitor e mouse, limitati da una rappresentazione 2D, con la realtà virtuale, è possibile esprimere e visualizzare anche la terza dimensione: la profondità. Perciò, questa tecnologia è particolarmente adatta alla visualizzazione di dati complessi come i volumi.

Dal lato software, è stato scelto Unity come ambiente di sviluppo. Si tratta di un software per la creazione di applicazioni 3D di tipo interattivo. Inoltre, si è fatto uso di librerie esterne per l'integrazione di *HTC Vive* con Unity (*VRTK*) e la visualizzazione dei dati volumetrici (*VolumeViewer)*.

Lo scopo dell'elaborato è quello di facilitare il lavoro di diagnosi, mettendo a disposizione un insieme di funzionalità che permettono al medico di: visualizzare i dati come se si trovassero nello spazio tridimensionale reale, manipolare il volume in modo da poter osservare una qualsiasi regione al suo interno mediante la dissezione; la possibilità di scalarlo, ruotalo e di marcare i punti d'interesse con pennarelli 3D.

Nei capitoli a seguire si discuterà:

1. la realtà virtuale, introducendo i concetti alla base di questa tecnologia. Verrà analizzato l'hardware dei visori moderni, spiegando come questi si differenziano dalle generazioni passate. Inoltre, si darà una panoramica degli utilizzi di tale tecnologia in settori diversi.
2. Le librerie e gli strumenti usati nello sviluppo di applicazione interattive per la realtà virtuale.

3. La teoria e funzionamento alla base delle moderne tecniche di diagnosi per immagini, approfondendo la tomografia computerizzata (TAC), la risonanza magnetica (RM) e i formati di memorizzazione digitali.
4. Le tecniche di visualizzazione tridimensionale di dati volumetrici: il rendering volumetrico. In particolare, verranno presentati tutti i passaggi e gli algoritmi necessari per ottenere un'immagine di questo tipo.
5. La progettazione e implementazione del progetto.

Indice

Elenco delle figure

Capitolo 1

Introduzione alla realtà virtuale

La realtà virtuale è una simulazione all'elaboratore di una situazione reale con la quale il soggetto umano può interagire, a volte per mezzo di interfacce non convenzionali, estremamente sofisticate, quali occhiali e caschi su cui viene rappresentata la scena e vengono riprodotti i suoni, e sistemi di interazioni, come guanti e controller, dotati di sensori per simulare stimoli tattili e per tradurre i movimenti in istruzioni per il software. Il fine della realtà virtuale è dare a chi la sperimenta l'impressione di trovarsi realmente immerso in un ambiente simulato per mezzo di tecnologie elettroniche. [1]

In questo capitolo verrà trattata la storia relativa allo sviluppo di visori per la realtà virtuale, ritenuta necessaria per comprendere il ruolo di questa tecnologia nel corso degli anni e delle numerose difficoltà ingegneristiche che ne hanno impedito una larga adozione. Successivamente verrà introdotto, sinteticamente, il sistema di funzionamento del visore *HTC Vive*, il quale rappresenta, attualmente, lo stato dell'arte nell'ambito consumer.

1.1 Breve storia

Il desiderio di immergersi completamente in un mondo parallelo è nato ben prima che il termine "realtà virtuale" venisse coniato. Infatti, già nel 1838, il fisico e inventore Charles Wheatstone ideò lo stereoscopio, il quale permette di visualizzare immagini con una illusione di tridimensionalità, ottenuta mediante due immagini leggermente traslate, divise da un separatore, in modo da simulare il punto di vista di entrambi gli occhi.

Su questo principio basilare, ma di fondamentale importanza, si basano anche i più sofisticati visori moderni.

1.1.1 Gli albori

I visori per la realtà virtuale sono anche chiamati HMD: dall'inglese head-mounted display, in quanto sono dispositivi che vengono indossati sulla testa e dotati di uno o più schermi.

Il primo HMD fu sviluppato da Ivan Sutherland nel 1968, denominato "La spada di Damocle" (figura 1.1) per via delle enormi dimensioni che richiedevano un supporto a soffitto. Si trattava di un sistema che era in grado di proiettare immagini di tipo wireframe, generate al computer, su lenti parzialmente trasparenti e per questo motivo è considerato anche il precursore alla tecnologia AR (realtà aumentata).

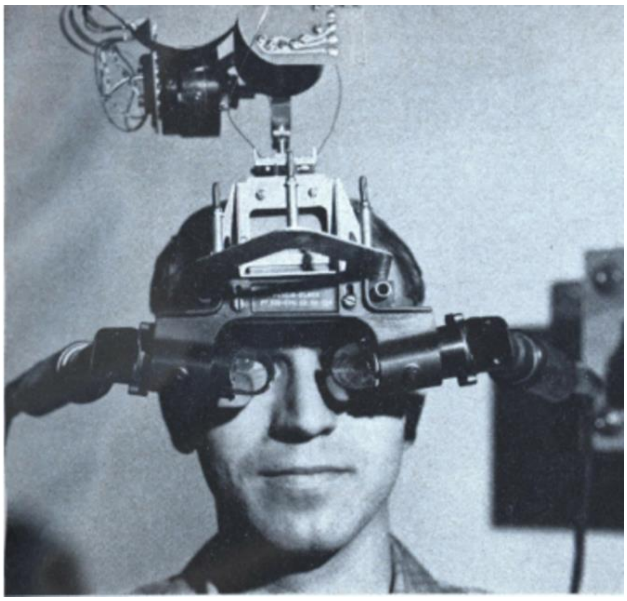


Figura 1.1 La spada di Damocle, Ivan Sutherland, 1968.

Figura 1.2 VIEW, HMD e dataglove sviluppato dalla NASA, 1985.

All'inizio degli anni '80, la NASA si interessò alla realtà virtuale per cercare di ridurre il costo di addestramento di missioni spaziali per gli astronauti. Così, nel 1985 svilupparono il *Virtual Interface Enviroment Workstation*, semplicemente riferito come VIEW (figura 1.2), questo era il primo HMD ad usare una grafica di tipo poligonale e la possibilità di interagire con l'ambiente tramite un guanto (il *dataglove*). Il *VIEW* diede un contributo essenziale allo sviluppo di tecnologie in ambito VR grazie a innovazioni quali l'audio 3D, display incorporato e il *dataglove*.

1.1.2 La prima generazione

Il termine *realtà virtuale* fu coniato nel 1987 da Jaron Lanier, fondatore di VPL, compagnia che collaborò con la NASA nello sviluppo del sistema *VIEW* e del *dataglove*, questo ha permesso ad essa di lanciare i propri prodotti sul mercato per primi, ma non impedì a loro di andare in bancarotta qualche anno più tardi.

Nel corso degli anni ‘90, la VR venne resa celebre dalla cultura popolare, in particolare da film come Tron (1982), The Lawnmower Man (1992) e Nirvana (1997), i quali ne ritraevano l’uso, spesso con hardware in commercio all’epoca, questo risultò in un grande interesse da parte del pubblico. Varie aziende, specialmente quelle di videogiochi, colsero l’opportunità per entrare in questo settore allo scopo di assicurarsi una fetta di un mercato potenzialmente enorme.

Una fra queste è SEGA, che presentò al CES (Consumer Eletronics Show) del 1991 un HMD dotato di tracking della posizione e audio 3D; originalmente pianificato per il lancio per la fine del ’93, il progetto fu successivamente cancellato per difficoltà incontrate durate lo sviluppo, in particolare i collaudatori lamentarono un forte mal di testa e nausea dopo l’uso [2].

Un altro esempio è il *Virtual Boy*, sviluppato da Nintendo nel 1995 e commercializzato ad un prezzo di $180, non era dotato di alcun tipo tracciamento, infatti non prevedeva nemmeno la possibilità di essere indossato, al contrario, era necessario appoggiarlo su un tavolo e posizionare la testa in prossimità alle lenti. Era dotato di uno schermo LCD con una risoluzione di 384x224 *pixels* e una *refresh rate* di 50 Hz, ma era in grado solamente di riprodurre 4 colori [3]. Unica peculiarità era quindi la grafica stereoscopica 3D; questo ha inevitabilmente generato un fallimento clamoroso per la casa produttrice, che ha dovuto ridurre il prezzo a soli $99 per evitare perdite maggiori.

Figura 1.4 DataVisor 80, $90'000, 1997.

Figura 1.3 Nintendo Virtual Boy, $180, 1995.

Nello stesso periodo vennero sviluppati anche visori di fascia alta, destinati ad un uso professionale. Uno dei più avanzati era il *N-Vision Datavisor 80* (1997) dotato di due schermi CRT con una risoluzione di 1280x1024, una elevata frequenza di aggiornamento di 180 Hz e un campo visivo di 120°; ma con un prezzo impressionante di $90'000, non era commercializzabile ad un pubblico ampio.

Dati i risultati deludenti dei prodotti disponibili sul mercato ad un prezzo ragionevole, verso la fine degli anni ’90, l’entusiasmo verso questa tecnologia è gradualmente svanito e nel primo decennio del 21esimo secolo, la VR era morta.

1.1.3 La rinascita della VR

L’interesse verso la realtà virtuale si riaccese quando nel 2010, il diciottenne Palmer Luckey, appassionato di elettronica e videogiochi, creò il primo prototipo dell’*Oculus Rift*.

L’approccio rivoluzionario di quest’ultimo è stato quello di sfruttare l’avanzamento tecnologico nel campo degli smartphone e tablet, oramai dotati di schermi ad alta risoluzione e ampia gamma di sensori integrati, con il risultato di ridurre drasticamente il costo del prodotto. Inoltre, grazie al supporto di John Carmack, creatore di *Doom*, Luckey riuscì a raccogliere 2,4 milioni di dollari nel 2012 attraverso una campagna di crowdfunding su Kickstarter per realizzare il prodotto.

Una sorprendente svolta avvenne nel 2014, quando Facebook annunciò di aver acquisito Oculus per 2 miliardi di dollari, a questo punto le altre aziende non volevano più rimanere a disparte a guardare, perciò appena un mese dopo, Sony rivelò un suo visore: il *PlayStationVR*, seguito da una partnership fra HTC e Valve per lo sviluppo del *Vive*. Questi HMD sono tutti dotati di tracciamento della posizione, schermi ad alta risoluzione con bassa latenza e la possibilità di interagire con il mondo virtuale attraverso controller di movimento addizionali.

Di fatto, l'*Oculus Rift* e il *HTC vive*, sono riusciti ad offrire una qualità simulativa pari o superiore ai più costosi sistemi VR della generazione passata ad un costo contenuto di $500 e $800 rispettivamente, invece che decine di migliaia.

1.2 Difficoltà ingegneristiche

Uno dei più grandi problemi che ostacolano un'ottimale esperienza in realtà virtuale è la cosiddetta "simulator sickness", ovvero il senso di nausea e malessere dovuto all'uso di un HMD.

Questo effetto collaterale si verifica quando si crea un conflitto di percezione fra i sensi, più comunemente fra l'apparato visivo e quello vestibolare, ad esempio quando ci si muove nella simulazione ma si è fermi nella realtà.

Per garantire un'esperienza ottimale, evitando il simulator sickness, è necessario che i visori adottino schermi idonei, in particolare sono state individuate le seguenti caratteristiche essenziali:

- **Alta risoluzione**, è stato stabilito come limite massimo, oltre alla quale non si è più in grado di distinguere la griglia formata dai pixel, una risoluzione di 60 pixel per grado visivo [4], ovvero circa 8K (7680x4320) per occhio se il campo visivo è di 110°.
- ***Refresh rate*** di almeno 75 Hz [5].
- ***Low persistence*** e global, lo schermo si accende solamente per mostrare il *framebuffer* per un 1-2 ms di tempo per poi spegnersi in attesa di un nuovo frame, inoltre global vuol dire che l'immagine viene mostrata per intero, eliminando l'effetto di *tearing*.
- **Latenza *motion-to-photon*** deve essere minore di 20 ms per indurre il senso di presenza all'utilizzatore [6]. Questo parametro è il tempo affinché il movimento effettuato venga

riflesso sullo schermo. Oltre a rovinare l'immersione, un'alta latenza può causare direttamente la *simulation sickness*.

Dal lato software, uno degli aspetti più interessanti e complessi è senza dubbio il rendering, in quanto è cruciale massimizzare l'equilibro fra la qualità visiva e le performance, onde evitare sconforto.

Il tipo di rendering usato in VR è lo stereo rendering che consiste nel produrre due immagini della stessa scena con una prospettiva leggermente differente, che rappresentano il punto di vista di entrambi gli occhi. Questo processo avviene in modo sequenziale, come viene mostrato in figura 1.5, dove viene prima elaborata l'immagine dell'occhio sinistro e poi quella dell'occhio destro, il V-Sync (sincronizzazione verticale) è attivo, in quanto un frame rate consistente favorisce il senso di fluidità dell'esperienza virtuale.

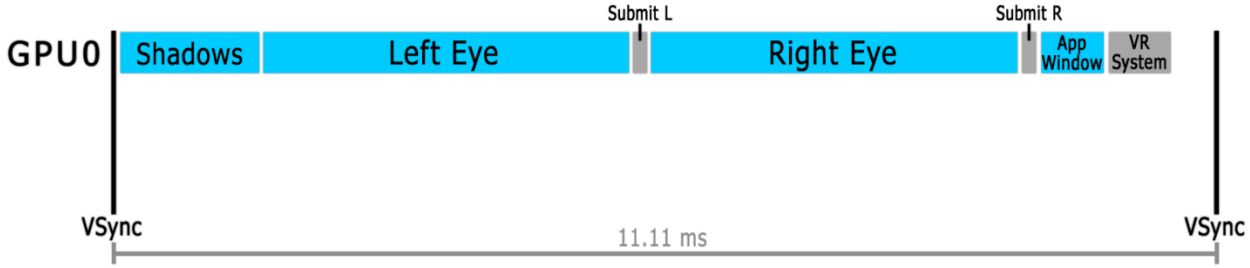


Figura 1.5 Rendering sequenziale su una singola GPU. Alex Vlachos, GDC 2015.

Essendo il rendering un problema di tipo *embarrassingly parallel*, ovvero di cui è relativamente facile costruire un algoritmo parallelo, è possibile usare più di una GPU per ridurre drasticamente i tempi di elaborazione, nel caso di due schede video (figura 1.6) è sufficiente assegnare ad ognuna di esse un'immagine da renderizzare.

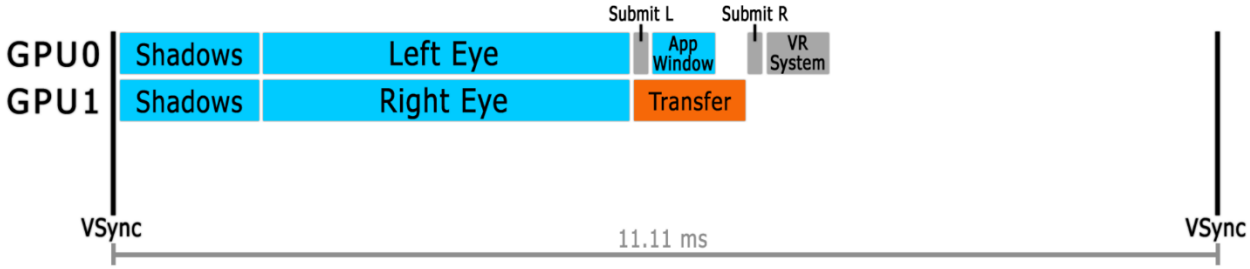


Figura 1.6 Rendering parallelo con 2 GPU. Alex Vlachos, GDC 2015.

La figura 1.7 mostra che le lenti distolgono la visione (di solito una distorsione a cuscinetto), prima di mostrare il framebuffer sullo schermo è necessario fare un passo aggiuntivo: bisogna applicare una distorsione cromatica/spaziale contraria (a barile) a quello delle lenti per poter visualizzare correttamente l'immagine. Da qui emerge un problema evidente, ovvero che alcune parti dell'immagine verranno ingrandite come la parte centrale, e invece le zone laterali rimpicciolite, risultando in una definizione minore al centro e uno spreco di risorse per le zone periferiche.

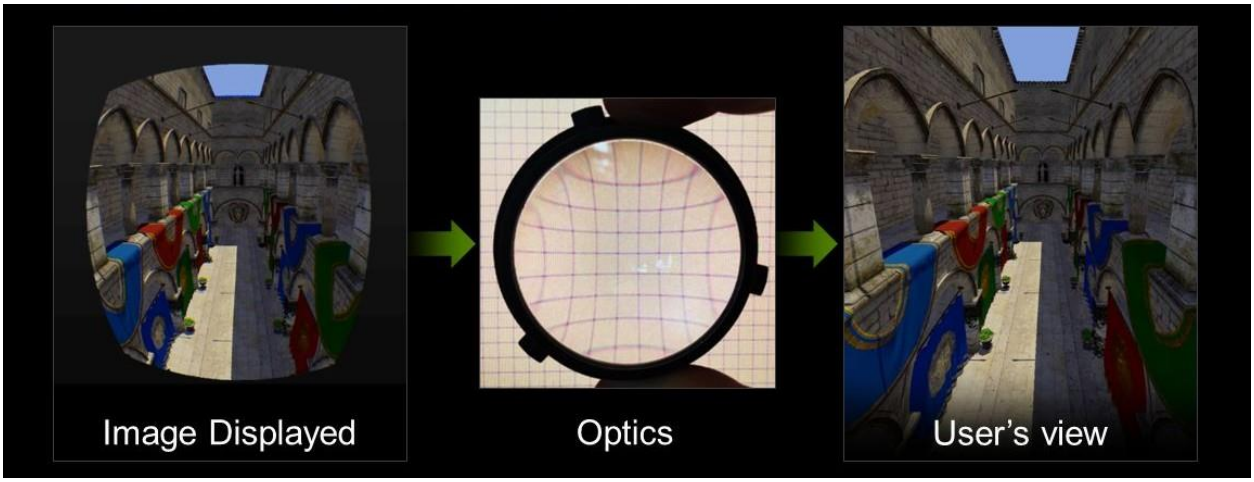


Figura 1.7 All'immagine originale viene applicata una distorsione a barile, che viene cancellato da una seconda distorsione a cuscinetto delle lenti per riprodurre un'immagine finale senza artefatti.

Una soluzione, a quest'ultima complicazione, è di applicare un render target di 1.4 volte maggiore alla risoluzione effettiva del display attraverso l'uso di una tecnica di anti-aliasing, detta *supersampling* [7]. Esistono, tuttavia, tecniche più efficienti come il *foveated rendering* che tramite l'aggiunta di un sensore in grado di tracciare il movimento del bulbo oculare, viene data la priorità alla parte dell'immagine che si sta osservando, riducendo la qualità nella zona circostante non a fuoco.

1.3 HTC Vive: in dettaglio

Il sistema VR, nato dalla collaborazione di HTC e Valve, rappresenta quello più avanzato attualmente disponibile sul mercato, grazie alle specifiche tecniche all'avanguardia e alla tecnica innovativa per il tracciamento.

Figura 1.8 HTC Vive e i suoi componenti: due controller di movimento e due base stations per il tracking.

Il visore è composto da due schermi AMOLED con matrice subpixel *PenTile* di tipo *global* e *low-persistence*, che quindi si accendono ciclicamente per un brevissimo lasso di tempo per evitare effetti di blur. La dimensione è di 3.6" ciascuno con una risoluzione di 1080x1200 pixel, per un totale di 2160x1200, infine la frequenza di aggiorna è di 90Hz [8].

Le lenti sono di tipo Fresnel, ovvero sono formate da numerose piccole curvature e rispetto a una lente sferica di equivalente potere diottrico sono molto più sottili, consentendo di raggiungere un campo visivo di 110°. Inoltre, hanno una messa a fuoco a infinito per non affaticare gli occhi.

L'HMD è anche dotato di sensori addizionali che forniscono informazioni aggiuntive al programmatore come l'accelerometro, il giroscopio e il sensore di prossimità, resi economici dall'avvento degli smartphone.

Sia il casco che i due controller di movimento inclusi sono dotati di sei gradi di libertà, rendendo possibile la "room scale VR", ossia la possibilità di muoversi liberamente all'interno di un'area di 4.5m x 4.5m e di interagire in modo naturale nel mondo virtuale, come illustrato nella figura 1.9.

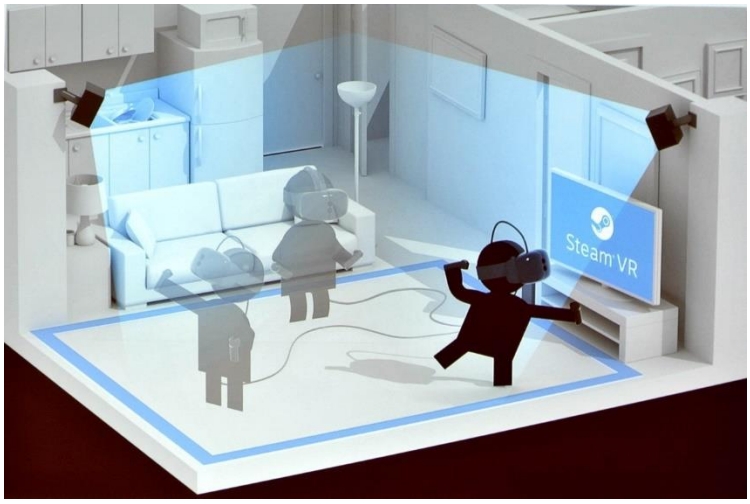


Figura 1.10 Room scale VR. Permette di muoversi liberamente in un'area di 4.5m x 4.5m.

La tecnologia di tracciamento 3D, denominato "Lighthouse", rende il tutto possibile. Oltre a conferire la facoltà di supportare un numero arbitrario di device tracciati simultaneamente, è dotato di un'ottima precisione, con un errore medio di circa 2mm [9]. Queste caratteristiche hanno convinto la NASA ad utilizzarlo per l'addestramento virtuale dei propri astronauti [10].

Figura 1.9 Una base station in funzione. Emette luce infrarossa che serve ai fotosensori per ricostruire la posizione.

L'idea alla base di questo sistema è molto ingegnosa: vengono posizionati due emettitori (base stations) agli angoli opposti dell'area d'azione, e hanno il compito di emettere un flash e un fascio di luce a infrarossi, figura 1.10. Il flash viene catturato da un array di sensori fotosensibili, posizionati sul dispositivo da tracciare, e rappresenta un segnale che fa partire un cronometro. Successivamente viene sparato un ampio fascio di luce a infrarossi che deve essere ricevuto da una

parte dei sensori, siccome la posizione di questi è predeterminata, una volta che si conosce **dov'è** posizionato quel fotosensore e **quando** viene colpito, è possibile calcolare la posizione dell'intero dispositivo mediante triangolazione. Inoltre, questo metodo conferisce anche il senso di orientamento senza ulteriori passaggi.

Figura 1.11 Un prototipo del Vive con i fotosensori scoperti.

Al contrario di tecniche basate su computer vision, non si hanno overhead significativi sulle performance anche nel caso di numerosi dispositivi, in quanto si tratta di calcoli poco dispendiosi per la CPU. Un difetto intrinseco del Lighthouse è costituito dalla presenza di componenti meccanici in movimento nelle base stations che col tempo si spostano, rendendo necessaria la ricalibrazione per un corretto tracciamento.

1.4 Applicazioni della Realtà Virtuale

La realtà virtuale trova ampio utilizzo in numerosi ambiti grazie alla possibilità di ricreare situazioni che sarebbero troppo pericolose, o economicamente non convenienti, nella realtà fisica. Diventa possibile, per esempio, esplorare una casa prima ancora che inizino i lavori di costruzione. Si pensi, inoltre, alle interazioni sociali, che non sono più limitate solamente da voce e video, ma è possibile sentire la presenza di altre persona intorno a sé tramite il linguaggio corporeo e le espressioni facciali. Storicamente, i principali utilizzatori della VR, sono l'esercito e gli enti di ricerca (NASA), che molto spesso sviluppano sistemi appositi. I simulatori di volo militari sono ampiamente usati per la loro efficacia; permettono ai piloti di provare nuove tecniche di formazione senza alcun pericolo. Una simulazione di un combattimento urbano in realtà virtuale, accelera l'addestramento delle reclute, perché li mette in prima persona davanti a situazioni di stress che appaiono reali.

Figura 1.12 Militari americani durante l'addestramento in VR.

Uno dei campi di applicazione dove la VR ha maggiore potenzialità è quello medico. Infatti, nella psicoterapia, ai pazienti affetti da disturbi post traumatici da stress (*Post Traumatic Stess Disorder*, *PTSD*), tra cui un gran numero di soldati ritornati dalla guerra in Iraq, viene fatto rivivere i momenti traumatici in un ambiente simulato, allo scopo di superare i traumi tramite l'esposizione a uno stress graduale e controllato. Inoltre, è usato nella chirurgia robotica (figura 1.13), dove il

chirurgo effettua l'operazione controllando un braccio meccanico tramite un joystick, in quanto è essenziale un feedback visivo e tattile molto preciso. Infine, la visualizzazione in realtà virtuale di immagini mediche a scopi diagnostici, fornisce un livello di dettaglio e di interazione ineguagliata, rendendo possibile la rappresentazione in scala 1:1 di TAC e RM e la manipolazione dei dati in modo naturale e intuitivo direttamente con il movimento delle mani. Perciò, la VR diventa uno strumento molto prezioso per il medico, in quanto facilita il processo di diagnosi, abbassando, di conseguenza, le probabilità di errori nell'interpretazione.

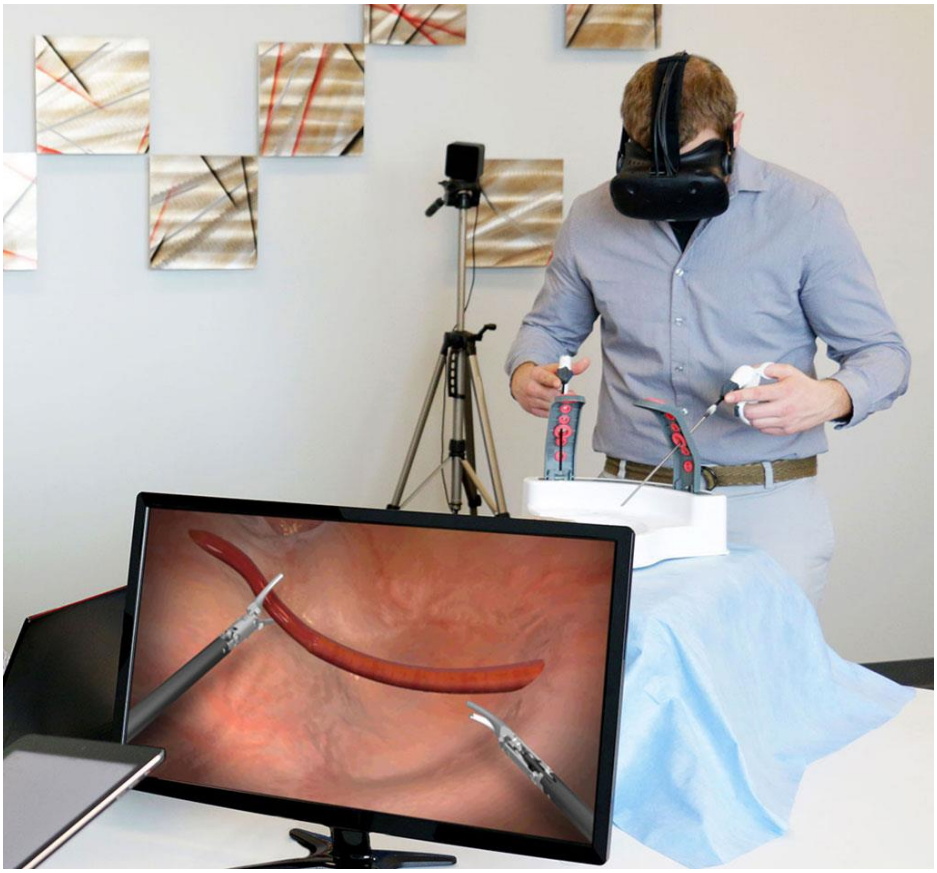


Figura 1.13 Simulazione di un intervento chirurgico in realtà virtuale con l'HTC Vive.

Capitolo 2

Tecnologie per lo sviluppo di applicazioni di realtà virtuale

In questo capitolo si analizzeranno le tecnologie usate per lo sviluppo del progetto, il motore di grafico Unity 3D, le librerie per lo sviluppo di applicazioni VR, e infine, il software open source Blender per la modellazione e texturing dei modelli poligonali.

2.1 Unity 3D

Unity è nato allo scopo di facilitare la progettazione di videogiochi e applicazioni interattive che fanno uso della computer grafica in tempo reale. Si tratta di un framework consolidato, che offre al programmatore un ambiente di sviluppo con un alto livello di astrazione, permettendogli di concentrarsi unicamente sulla logica dell'applicazione.

Il software è disponibile in tre tipi di licenze: la *Personal* è completamente gratuita, a patto che gli introiti derivati dalle vendite siano minori di $100'000; una volta raggiunto queste limite, sarà necessario sottoscrivere un abbonamento mensile per la versione *Plus,* o la più costosa *Pro*.

I motivi che hanno portato alla scelta di usare Unity per realizzare questo progetto di tesi sono molteplici e sono da ricercarsi, in primo luogo, nell'incredibile integrazione delle ultime tecnologie grafiche rilasciate dai produttori di hardware, come Vulkan, iOS Metal, DirectX12, nVidia VRWorks e AMD LiquidVR [11]; in secondo luogo, nella velocità con cui è possibile eseguire test, si riduce il tempo di prototipazione, e di conseguenza, ciò permette di iterare il progetto in maniera molto rapida.

Inoltre, Unity è estremamente popolare nell'ambito videoludico, infatti, oltre ai principali sistemi operativi su pc (Linux, Windows, Mac Os), supporta anche tutte le console da gioco, gli

smartphone, le smart tv, i browser (WebGL), e tutti i visori che fanno uso di OpenVR e Oculus SDK, per un supporto totale di oltre 25 piattaforme [12].

2.1.1 L'interfaccia grafica di Unity

Al contrario dei motori grafici basati su libreria, in Unity è presente un'interfaccia grafica che permette di eseguire azioni comuni, come movimento di oggetti nella scena, texturing, animazione, e di vederne subito l'effetto tramite la finestra di preview, senza dover scrivere e compilare codice.

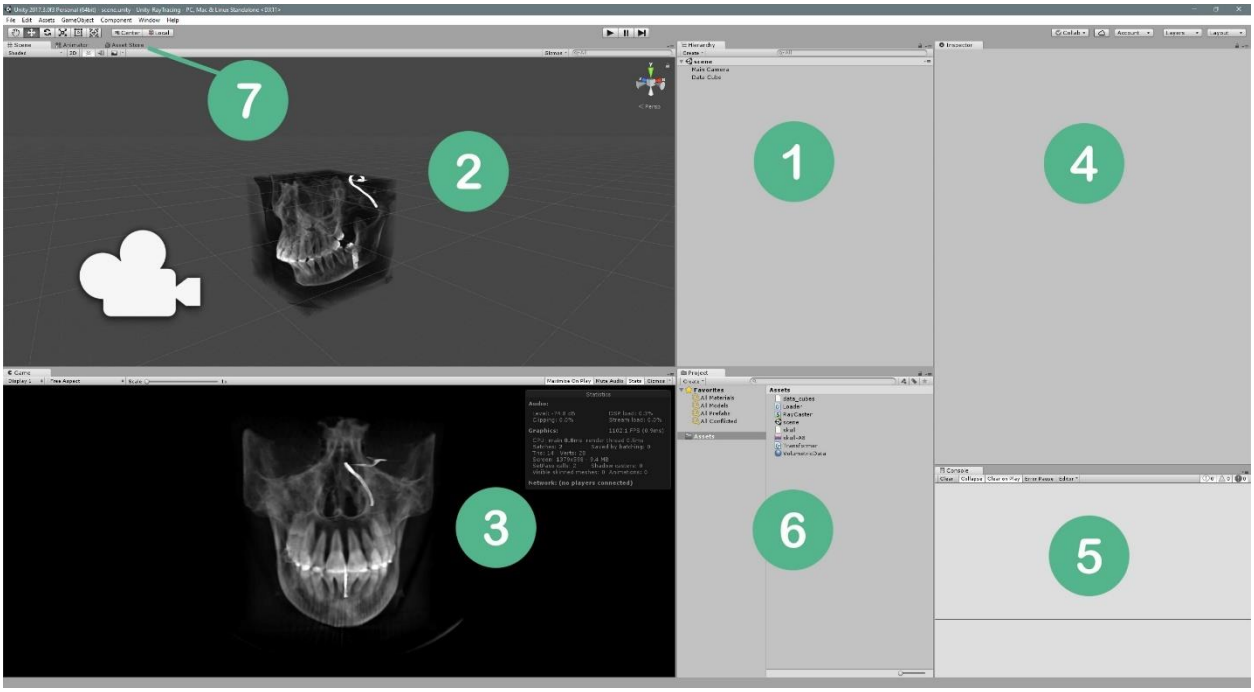


Figura 2.1 L'interfaccia grafica di Unity, sono state evidenziate le principali finestre

La figura 2.1 mette in evidenza i punti salienti dell'interfaccia dell'editor:

1. **Hierarchy**: questa finestra elenca tutti gli oggetti presenti nella scena corrente; questi possono includere, per esempio, forme geometriche, modelli poligonali, luci e telecamere. Inoltre, permette di visualizzare la gerarchia tra gli oggetti; per esempio, un oggetto può avere uno o più oggetti figli.
2. **Scene**: mostra il contenuto della scena in modo grafico, aggiornando in tempo reale le modifiche apportate agli elementi presenti.

3. **Game**: rende possibile la visualizzazione della scena come apparirebbe una volta lanciato il gioco o l'applicazione.
4. **Inspector**: mostra le informazioni e le proprietà dell'oggetto selezionato.
5. **Console**: stampa messaggi di warning o di errore da parte di Unity, e codice di debug inserito dall'utente.
6. **Project**: finestra che include tutti gli asset/risorse che possono essere usati nel progetto.
7. **Asset store**: portale nel quale è possibile reperire gli asset, che possono essere a pagamento o gratuiti.

2.1.2 GameObjects e il component pattern

Unity fa ampio ricorso agli oggetti (GameObjects). Tutti gli elementi inseribili nella scena, dai modelli poligonali alle luci, camere ed effetti speciali sono definiti come GameObjects. Questi, da soli non possono fare niente, affinché vengano attribuite delle azioni, è necessario aggiungere ad essi dei componenti. Ogni GameObject ha un componente di default, non rimovibile, chiamato *Transform* [13], grazie al quale è possibile modificarne la posizione, rotazione e la dimensione nei tre assi tridimensionali; inoltre, è anche possibile costruire componenti customizzati facendo uso di script per determinarne il comportamento.

L'uso dei componenti è reso possibile dal pattern *Component*, integrato in Unity, che nasce dalla necessità di separare gli interessi, ad esempio per evitare che il codice del rendering sia in qualche modo dipendente da quello per la fisica o l'animazione. L'obiettivo è, quindi, quello di conferire una maggiore flessibilità per lo sviluppo di funzioni e semplificarlo tramite componenti riusabili su più oggetti.

Gli oggetti nella scena possono essere duplicati e usati più volte, ma si verifica una complicazione quando si decide di modificare una proprietà di un componente, ovvero questa dovrà essere ripetuta su tutti gli oggetti copiati. Fortunatamente, in Unity è possibile usare un tipo di asset chiamato **Prefab** (o prefabbricato), che permette di creare istanze di un oggetto predefinito, in modo tale che qualunque modifica ad uno di questi oggetti venga riflesso in tutti quelli appartenenti allo stesso prefabbricato.

2.1.3 Scripting in Unity

Per definire la logica dell'applicazione, il comportamento degli oggetti e dell'input, occorre creare degli script da assegnare ai singoli componenti. Unity offre due linguaggi di programmazione per lo sviluppo di scripts:

- **UnityScript**, derivato da JavaScript e ideato esclusivamente per Unity.
- **C#**, rappresenta lo standard de facto, in quanto offre una libreria molto più matura ed una migliore ottimizzazione.

Ogni script deve estendere la classe base MonoBehaviour, che fornisce i seguenti metodi:

- **Start()** : chiamato solo una volta quando lo script è abilitato, prima di Update(), è usato principalmente per inizializzare la classe.
- **Update()** : viene chiamato ad ogni frame se lo script è abilitato, è usato per implementare il comportamento.
- **FixedUpdate()** : simile ad Update(), ma l'invocazione è sincronizzata con il *framerate*. È utile quando si lavora con la fisica.
- **LateUpdate()** : chiamato ad ogni frame, dopo che tutti i tipi di chiamate Update() hanno terminato.
- **OnGUI()** : chiamato ad ogni evento riguardante l'interfaccia grafica.
- **OnDisable()** : chiamato quando lo script diventa inattivo o disabilitato.
- **OnEnable()** : chiamato quando lo script viene abilitato.

A questo punto, per ottenere il comportamento desiderato, basterà effettuare l'override sui metodi interessati e inserire le istruzioni corrette. Tuttavia, è importante menzionare l'ordine di esecuzione delle funzioni evento durante il ciclo di vita di uno script, mostrato in figura 2.2.

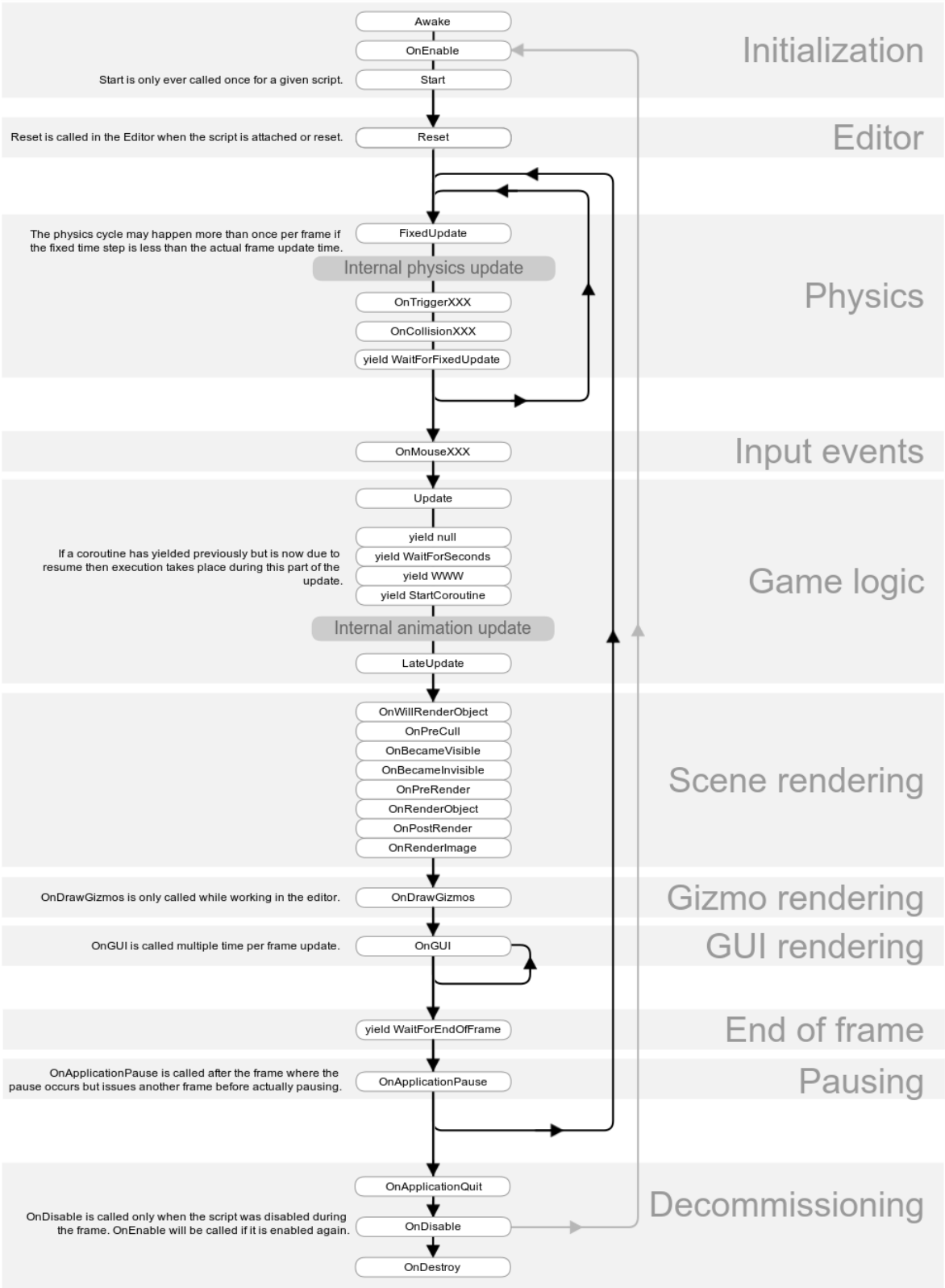


Figura 2.2 Diagramma che mostra il ciclo di vita di uno script [14]

L’ordine di esecuzione delle funzioni è particolarmente importante perché permette di conoscere quando il codice sviluppato verrà effettivamente eseguito da Unity; ad esempio, tutti i metodi riguardanti la fisica vengono chiamati prima di quelli dell’input, segue l’aggiornamento della logica dell’applicazione, e infine il rendering della scena.

Quando una funzione viene chiamata, Unity aspetta che questa termini prima di proseguire con un’altra. Questo significa che tutte le istruzioni contenute all’interno di una funzione devono essere eseguite all’interno di un periodo di tempo, pari alla frequenza di aggiornamento del frame. Una soluzione a questo inconveniente è quello di usare le ***Coroutines***, un meccanismo che consente di mettere in pausa una funzione, e di riprenderne l’esecuzione in un secondo momento. Di seguito viene mostrato la struttura di uno script:

```
1.  public class ChangeColor : MonoBehaviour {
2.
3.      private Renderer rend;
4.
5.      // Use this for initialization
6.      void Start () {
7.          rend = GetComponent<Renderer>();
8.      }
9.
10.     // Update is called once per frame
11.     void Update()
12.     {
13.         if (Input.GetKeyDown("f"))
14.         {
15.             StartCoroutine("Fade");
16.         }
17.     }
18.
19.     IEnumerator Fade()
20.     {
21.         for (float f = 1f; f >= 0; f -= 0.1f)
22.         {
23.             Color c = this.rend.material.color;
24.             c.a = f;
25.             this.rend.material.color = c;
26.             yield return new WaitForSeconds(.1f);
27.         }
28.     }
29. }
```

Questo script ha lo scopo di sbiadire il colore dell’oggetto su cui è applicato, il for loop all’interno della funzione Fade() viene eseguito nell’arco di più frame, grazie all’uso di una *Coroutine*.

2.1.4 Illuminazione in Unity

Storicamente, Unity usa un modello di illuminazione locale di tipo real-time. Questo significa che la luce viene aggiornata ad ogni frame, ma solo i raggi che colpiscono direttamente l'oggetto contribuiscono alla sua illuminazione. Di conseguenza, viene usato una luce ambientale per evitare le parti non colpite siano troppo buie, in quanto i raggi non sono in grado di ribalzare come accade nel mondo reale.

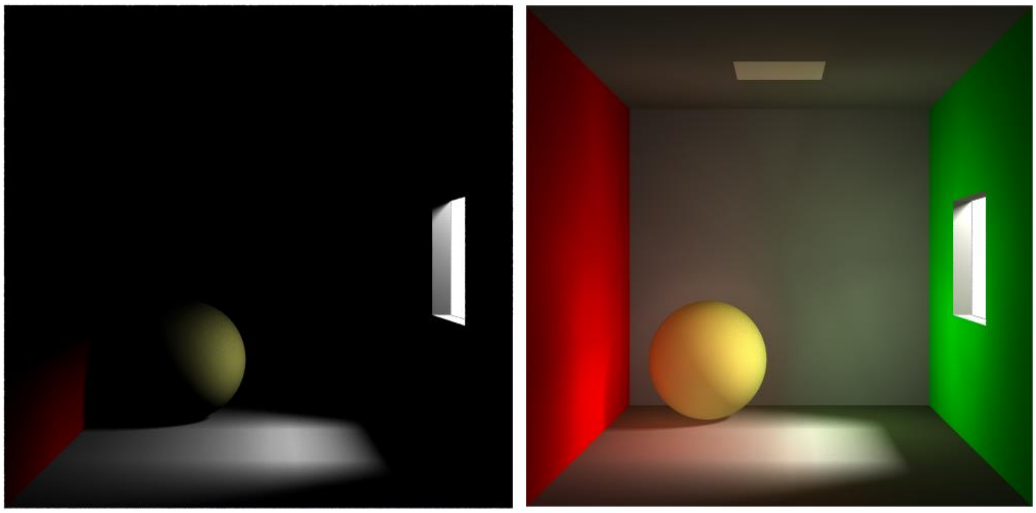


Figura 2.3 Differenze fra illuminazione locale/realtime (a sinistra) e illuminazione globale (a destra).

Dalla versione 5.0, rilasciata nel 2015, Unity ha introdotto il supporto a una nuova tecnica chiamata *Precumputed Realtime GI [15]*. Si tratta di un modello di illuminazione globale precalcolato, ossia, al momento della compilazione vengono calcolati tutti i possibili rimbalzi che le luci possono generare su un oggetto e vengono salvate per l'uso durante l'esecuzione. Questo permette di modificare il numero e la tipologia di luce nella scena, la loro posizione, orientamento, e la luce indiretta verrà aggiornata di conseguenza. Tuttavia, questo metodo è applicabile solo ad oggetti statici.

2.1.5 Materiali in Unity

I materiali sono usati per cambiare l'aspetto di un modello poligonale; per esempio, per simulare l'apparenza di una finestra, sarà necessario assegnare un materiale speculare semitrasparente. In Unity i materiali sono formati da:

- **Shaders**, sono dei piccoli script e determinano come il colore di ogni pixel cambia in relazione all'illuminazione e ai parametri di configurazione.
- **Textures**, sono immagini bitmap e rappresentano aspetti superficiali del materiale, come il colore, i riflessi e la rugosità.

Un materiale specifica quale shader usare (uno solo), e lo shader determina quali sono i parametri disponibili, tra quali anche il numero e il tipo di textures. Unity mette a disposizione una serie di shaders pronti per l'utilizzo, ma è anche possibile sviluppare shader personalizzati per ottenere effetti particolari; per esempio, visione a raggi x, effetti particellari e liquidi.

2.2 Librerie

Per interagire con l'hardware, nel caso del progetto con *HTC Vive* bisogna usare delle librerie. In questa sezione si presenteranno le principali librerie disponibili agli sviluppatori, nonché usate nel progetto.

2.2.1 OpenVR

Le applicazioni per l'*HTC Vive* sono sviluppate usando una API (application programming interface) chiamata OpenVR. Si tratta di un'interfaccia di programmazione ideata da Valve con l'obiettivo di fornire allo sviluppatore un modo unico per accedere all'hardware, senza discriminazione di modello e marca. In particolare, essa offre una serie di metodi standard che garantiscono un set di funzionalità su tutti i device supportati; spetterà quindi, ai produttori di hardware adeguarsi a questo standard, implementando un driver compatibile con essa.

L'API è scritta in C++, ed è un insieme di file header con funzioni virtuali, che forniscono solamente l'interfaccia; essenzialmente, è un "ponte" fra lo sviluppatore e l'SDK vero e proprio. Le principali interfacce sono sei [16]:

- **IVRSystem**: interfaccia primaria per accedere al display, distorsione dell'immagine da effettuare, tracciamento, controller ed eventi.
- **IVRChaperone**: fornisce l'accesso al sistema *Chaperone*, che costituisce il perimetro di utilizzo della tecnologia room scale VR.
- **IVRCompositor**: permette di effettuare il rendering 3D tramite il componente *Compositor*, che è un'astrazione che consente di semplificare il processo di visualizzazione sullo schermo del visore, occupandosi della distorsione, sincronizzazione e altre problematiche.
- **IVROverlay**: permette di effettuare il rendering 2D tramite il *Compositor*.
- **IVRRenderModels**: permette di accedere ai modelli poligonali dell'hardware.
- **IVRScreenshots**: permette all'applicazione di effettuare screenshots dello schermo.

Una precisazione è dovuta: l'appellativo "open" in OpenVR non significa open source, ma indica la liberà, da parte dei produttori, di implementare l'API senza costi.

2.2.2 SteamVR

La principale implementazione di OpenVR è SteamVR, sempre sviluppato da Valve, è il layer di basso livello che implementa l'API e che realmente interagisce con l'hardware. Quindi include i driver specifici per i visori supportati, che al momento sono: HTC Vive, Oculus Rift e i visori Windows Mixed Reality.

SteamVR è, quindi, un programma che rimane in esecuzione durante tutto il tempo in cui si usa il visore, e fornisce una completa integrazione alla piattaforma di gaming Steam con un'interfaccia di navigazione apposita, che include un'ambiente tridimensionale personalizzato, la quale funge da home quando non ci sono programmi attivi. Inoltre, tramite SteamVR avviene la configurazione del sistema C*haperone*, che consiste in un overlay a forma di griglia che compare quando ci si avvicina troppo, fisicamente, ai limiti del perimetro d'uso.

Altre funzionalità offerte sono:

- ***Supersampling***: renderizzare a una risoluzione maggiore di quella standard, per ottenere immagine più nitide.
- **Grafico** delle performance in tempo reale.
- ***Theather mode***: avvia i giochi e le applicazioni di Steam che non hanno supporto nativo alla realtà virtuale.

2.2.3 SteamVR Unity plugin

L'integrazione di SteamVR con Unity è resa possibile con l'omonimo plugin (figura 2.4), disponibile sull'asset store. Questo package contiene il necessario per lo sviluppo di applicazioni per i dispositivi VR supportati.

I componenti più importanti del package sono costituiti da due oggetti prefabs, e sono:

- [**CameraRig**], per manipolare ed accedere all'HMD e ai controller. Questa sostituisce la camera di default di Unity.
- [**SteamVR**], si occupa di integrare SteamVR con tutte le funzionalità discusse precedentemente.

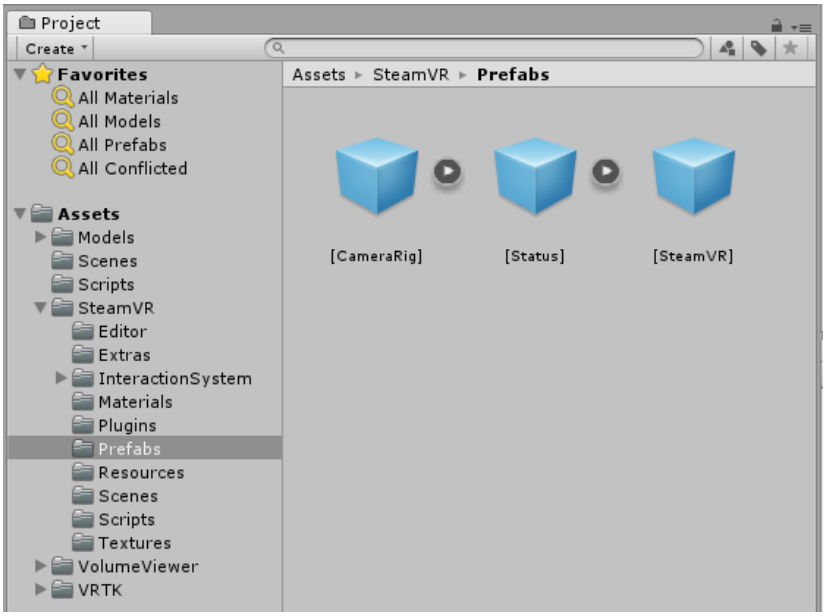


Figura 2.4 SteamVR plugin importato in Unity.

Il prefab [CameraRig] è formato da una gerarchia di GameObjects. Essa è costituita dagli oggetti: *Controller(left)*, *Controller(right)* e *Camera(head)*. Questi oggetti figli hanno tutti un componente script chiamato *SteamVR_TrackedObject*, che fornisce la posizione tracciata dai rispettivi device.

Purtroppo, la documentazione di questo plugin è scarsa. Inoltre, offre solamente un insieme di feature basilari, e di conseguenza, richiede un tempo di sviluppo elevato in quanto si dovrà sviluppare da zero anche le meccaniche di interazione più triviali.

2.2.4 La libreria VRTK

La libreria Virtual Reality Toolkit (VRTK), disponibile sotto forma di plugin gratuito sull'asset store di Unity, è un wrapper di diversi SDK, tra cui SteamVR, Oculus, Ximmerse e Daydream. Si tratta di progetto che espande SteamVR, in termini di funzionalità, semplificando notevolmente il processo di sviluppo.

In particolare, VRTK mette a disposizione una serie di soluzioni per interagire con il mondo virtuale [17], alcune di queste sono:

- Sistema di locomozione per muoversi nello spazio.
- Interazione con l'ambiente, ad esempio, per toccare, afferrare e usare oggetti.

- Interagire con l'interfaccia utente di Unity.
- Fisica degli oggetti.

L'implementazione delle funzionalità è molto intuitiva, ad esempio nel caso si volesse rendere un oggetto afferrabile: sarà sufficiente aggiungere il componente script, chiamato *VRTK_InteractableObject*, e attivare l'opzione *"is Grabbable"* (figura 2.5). Inoltre, è possibile modificare il comportamento di questi script tramite il meccanismo dell'ereditarietà, eseguendo l'override dei metodi opportuni.

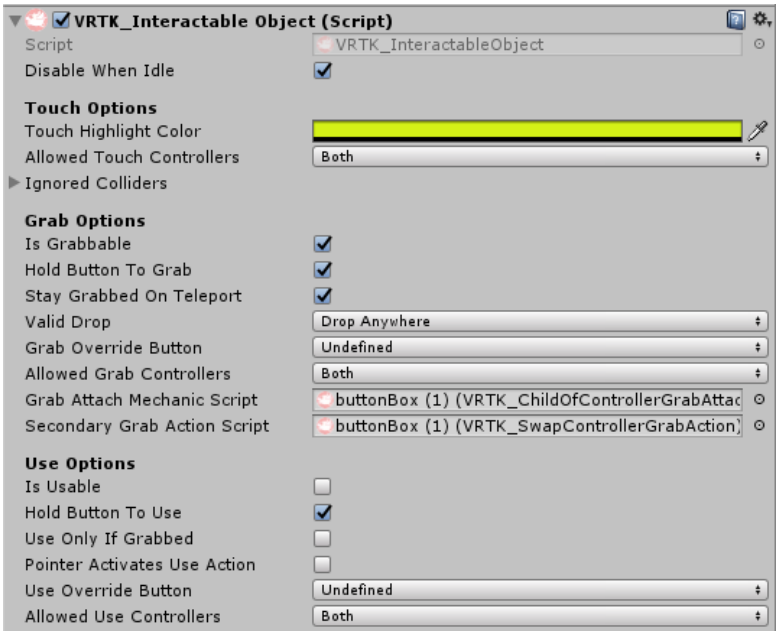


Figura 2.5 Opzioni presenti nella finestra inspector, dello script della libreria VRTK che abilita l'interazione dell'oggetto a cui è assegnato.

Ulteriore motivo, per cui è stato scelto di usare questa libreria, è rappresentato dalla grande quantità di risorse reperibili; infatti oltre alla documentazione ufficiale e ai video tutorial disponibili, all'interno del package è inclusa anche un'ampia collezione di esempi che ricoprono gran parte delle funzionalità.

2.3 Blender

Blender è un software open source per la creazione di contenuti 3D, permette la modellazione, il rigging, texturing, animazione, simulazione di fluidi e particelle, video editing e compositing. Trattandosi di un software libero ha attratto un'ampia comunità di sviluppatori, di conseguenza, il programma è in continuo aggiornamento con nuove funzionalità; al momento l'ultima versione è la 2.80, che ha introdotto un nuovo motore di rendering realtime chiamato Eevee. L'insieme di queste caratteristiche lo hanno reso estremamente popolare sia tra i dilettanti che tra i professionisti del settore, tanto da riuscire a competere a testa alta con software proprietari ad alto budget come Maya e 3D Studio Max di Autodesk.

Nel contesto di questo progetto, è stata naturale la scelta di usare Blender per la modellazione degli ambienti virtuali grazie all'integrazione con Unity, infatti quest'ultimo è in grado di leggere in modo nativo i file salvati da Blender con estensione *.blend*, permettendo quindi di inserire i modello nella scena semplicemente trascinandoli nella finestra *Hierarchy*.

Capitolo 3

Diagnostica per immagini

Dopo aver discusso ampiamente della realtà virtuale, si ritiene opportuno introdurre il dominio dell'applicazione sviluppata a corredo di questa tesi. Perciò, questo capitolo verrà dedicato alla diagnostica per immagini, partendo da una sintetica introduzione ai concetti base, per poi approfondire la tomografia computerizzata, la risonanza magnetica, e infine gli standard relativi ai formati digitali.

3.1 Introduzione

La diagnostica per immagini, o imaging, indica tutte le modalità di formazione delle immagini biomediche utilizzate a scopo diagnostico e, in alcuni casi, terapeutico. Il requisito fondamentale è una forma di energia, che, per produrre l'immagine, deve essere in grado di penetrare i tessuti [18]. Per l'imaging in campo medico, viene usato lo spettro elettromagnetico fuori dalla luce visibile, che include: i raggi-x per le mammografie e la tomografia computerizzata, radiofrequenze nella risonanza magnetica, e i raggi gamma nella medicina nucleare.

Il principio fondamentale alla base di questa disciplina, ad eccezione della medicina nucleare, risiede nell'interazione fra l'energia fornita e l'organo di cui si vuole ottenere un'immagine. Se l'energia passasse attraverso il corpo inalterata, senza nessuna informazione riguardante la composizione interna, allora, di conseguenza, sarebbe impossibile ricavarne un'immagine. La medicina nucleare, invece, prevede l'inserimento di una sostanza radioattiva all'interno del paziente, che interagendo con i raggi gamma, vengono ricavate le informazioni necessarie per creare l'immagine.

I fenomeni fisici che si verificano durante l'interazione dell'energia con i tessuti sono riflessione, rifrazione e assorbimento, che variano in base al tipo di tessuto incontrato; ad esempio, le ossa, in quanto più dense, assorbiranno una quantità maggiore di energia, rispetto alla pelle esterna, rendendo possibile la loro distinzione. Generalmente, per ottenere una qualità dell'immagine migliore, si dovrà sottomettere il paziente ad una dose maggiore di energia. Ovviamente, è necessario trovare il giusto compromesso fra la sicurezza per il paziente e la qualità dell'immagine.

3.1.1 Le modalità

Come accennato precedentemente, le diverse tipologie di immagini sono ottenute al variare del tipo di energia e alla tecnologia di acquisizione utilizzata. Le modalità della diagnostica per immagini sono:

- **Radiografia**: scoperta nel 1895 dal fisico Wilhelm Roentgen, è eseguita per mezzo di una sorgente di raggi-x, posta ad un lato del paziente e, un rilevatore dall'altro. Consiste in un impulso di raggi-x, della durata di una frazione di secondo. Parte dei raggi colpisce il paziente e, di conseguenza, vengono attenuati (per assorbimento o rifrazione) a seconda del tipo di tessuto, risultando in una distribuzione omogenea di raggi-x che costituisce l'immagine finale [18].
- **Fluoroscopia**: si tratta di un tipo di acquisizione di immagini a raggi-x in tempo reale, allo scopo di creare un filmato.
- **Mammografia**: è una radiografia ottimizzata per il seno, caratterizzata da una dose radioattiva molto più bassa di quella normale. Viene effettuata per identificare tumori e cisti [19].
- **Tomografia computerizzata (TAC)**: si diffuse negli anni '70 e costituisce la prima modalità di imaging resa possibile dal computer. Il termine tomografia si riferisce a un'immagine (grafia) di una sezione o fetta (tomo). Le TAC sono prodotte dal passaggio, attraverso il corpo del paziente, di raggi-x ad angoli diversi, ruotando la sorgente di raggi-x opportunamente intorno al corpo. Le informazioni collezionate in questo modo, vengono elaborate al computer attraverso un algoritmo che ricostruisce i dati, ottenendo un volume [20]. Il vantaggio principale, di questa modalità rispetto alla radiografia tradizionale, risiede

nell'abilità di mostrare immagini tridimensionali delle sezioni, presentando una visione delle parti interessate senza sovrapposizioni di elementi esterni.

- **Risonanza magnetica**: vengono usati degli scanner che generano campi elettromagnetici da 10'000 a 60'000 volte maggiori di quello terrestre. Si basa su un fenomeno fisico, chiamato appunto "risonanza magnetica nucleare delle proprietà del protone dell'idrogeno", il quale si trova abbondantemente nei tessuti biologici (acqua). Il protone, quando sottoposto all'azione di un intenso campo magnetico (da 1,5 a 3 Tesla) e irradiata da un'onda radio, riemette l'energia delle onde radio che vengono rilevate da un'antenna; il sistema RM, quindi, usa la frequenza magnetica e la fase delle radiofrequenze catturate per determinare la posizione di ciascun segnale dal paziente [20]. Come per le TAC, la RM produce un insieme di immagini tomografiche, ma ha il vantaggio di avere un contrasto migliore, il che la rende particolarmente adatta per diagnosticare i tessuti molli, come il cervello. Inoltre, non presenta effetti indesiderati, ma a differenza dei pochi secondi richiesta per una TAC, la RM necessita decine di minuti, perciò non è adatta in tutti i casi che comportano movimento.
- **Ecografia**: sfrutta l'energia meccanica formata dalle frequenze alte del suono, detta a ultrasuoni. La sorgente deve essere in contatto sulla zona d'indagine, viene usato un trasduttore che emana brevi impulsi di ultrasuoni, questi viaggiano all'interno del corpo e vengono riflessi dagli organi interni [21]. Questa tecnologia viene applicata solo in alcune aree del corpo, in quanto è molto sensibile a interferenze; per esempio, dovute alle cavità d'aria o all'eco generato dalle ossa.
- **Medicina nucleare**: a differenza delle tecniche elencate finora per costruire l'immagine, nella medicina nucleare la fonte d'energia, risiede all'interno del corpo. Perciò, una sostanza chimica o un isotopo radioattivo deve essere inserito nel paziente, per via orale o per iniezione [22]. Le principali tecniche di questa tipologia sono: Imaging planare, ovvero una mappa 2D della distribuzione dei radioisotopi; Tomografia a emissione di fotone singolo (SPECT), rappresenta la controparte della TAC; Tomografia a emissione di positroni (PET), è quella più avanzata e costosa, in quanto in grado di fornire informazioni di tipo fisiologico, permettendo di ottenere mappe dei processi funzionali.

3.1.2 Le proprietà dell'immagine

Le proprietà principali, nel campo della diagnosi per immagini, sono il contrasto e la risoluzione. La prima indica la differenza nei livelli della scala di grigio dell'immagine, una distribuzione uniforme non presenta contrasto, mentre un'immagine con una transizione netta fra grigio chiaro e grigio scuro ha un elevato contrasto.

Per i raggi-x, il contrasto è prodotto dalle differenze nella composizione dei tessuti, che determina il coefficiente di assorbimento di raggi-x, il quale è dipendente dalla densità (g/cm^3) e dalla carica nucleare efficace (Z_{eff}); per esempio, le ossa producono un alto contrasto, in quanto hanno una carica (nucleare) efficace ($Z_{eff} = 20$) molto diversa da quella dei tessuti ($Z_{eff} = 7$), grazie all'alta concentrazione di calcio (Z = 20) e fosforo (Z = 15).

Il contrasto nella RM è principalmente determinato dalla densità dei protoni e dal tempo di rilassamento (quanto velocemente i protoni rilasciano l'energia assorbita). La densità di protoni varia in relazione al tipo di tessuto, per esempio, è presente un'alta concentrazione di idrogeno (e quindi protoni) nei tessuti grassi. Inoltre, è possibile manipolare il tempo di rilassamento, modificando la sequenza delle radiofrequenze e il campo magnetico [20].

La risoluzione fornisce il grado di dettaglio ottenuto, si riferisce all'abilità di individuare la presenza di oggetti di piccole dimensioni. Nella tabella 3.1 viene fornito un panorama.

Tabella 3.1 Risoluzione delle varie modalità per la diagnostica per immagini

Risoluzione	Risoluzione (mm)
Radiografia digitale	0.17
Fluoroscopia	0.125
Mammografia digitale	0.05-0.10
Tomografia computerizzata	0.3
SPECT	7
PET	5
RM	1
Ultrasuoni	0.3

3.2 Tomografia computerizzata

In questa sezione si andranno ad esaminare i principi fisici, le modalità di acquisizione e ricostruzione delle immagini della tomografia computerizzata.

3.2.1 Principi di interazione dei raggi-x

Alla base della tomografia computerizzata vengono usati i raggi-x, e quindi si è ritenuto opportuno, in questo paragrafo, spiegare come questi interagiscono con la materia.

Quando i raggi-x attraversano la materia, i fotoni possono penetrare senza interazione, con diffusione (scattering) o assorbiti. Le interazioni di interesse per la radiologia si classificano in:

- **Scattering di Rayleigh**, accade quando l'interazione è fra un fotone ed un elettrone interno dell'atomo [23], principalmente se i raggi hanno una bassa energia (come per le mammografie). L'energia produce una oscillazione degli elettroni, che a sua volta la rilascia, emettendo un fotone con la stessa energia ma in direzione leggermente diversa (figura 3.1). In quanto gli elettroni non vengono espulsi, non avviene il fenomeno della ionizzazione.

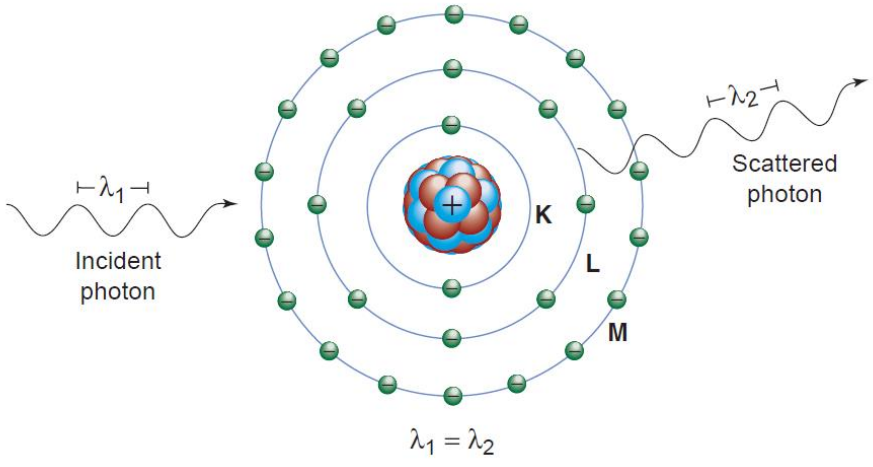


Figura 3.1 Diffusione di Rayleigh: un fotone si scontra con un elettrone, producendo un'oscillazione che crea un nuovo fotone con la stessa energia, ma direzione diversa.

Le probabilità che si verifichi questo effetto sono abbastanza basse, infatti nei tessuti molli, lo scattering di Rayleigh costituisce solamente il 5% delle interazioni totali per raggi con energia superiore a 70 keV (Elettronvolt), e il 10% nel caso di 30 keV.

- **Effetto Compton**, si verifica quando il fotone colpisce un elettrone compreso nel guscio esterno dell'atomo. L'elettrone, di conseguenza, viene espulso (ionizzazione) e il fotone viene deviato con un'energia ridotta, questo processo è illustrato nella figura 3.2. Inoltre, come per tutte le interazioni, sia il momento che l'energia totale deve essere conservata. Perciò, l'energia del fotone originale (E_0) è uguale all'energia del fotone deviato (E_{sc}) sommata all'energia cinetica dell'elettrone espulso (E_{e-}) [24]; ottenendo la seguente formula:

$$E_0 = E_{sc} + E_{e-}$$

L'energia del fotone deviato può essere calcolata usando l'angolo di incidenza θ:

$$Esc = \frac{Eo}{1 + \frac{Eo}{511keV}(1 - cos\theta)}$$

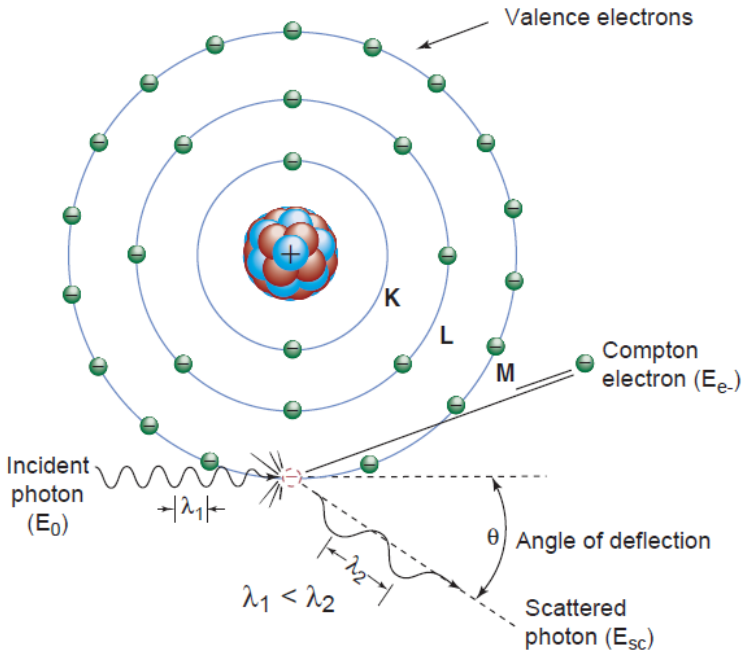


Figura 3.2 Effetto di compton: il fotone colpisce un elettrone compreso nel guscio dell'atomo, espellendolo. Si crea un nuovo fotone con energia inferiore e una deviazione pari all'angolo ***θ****.*

- **Effetto fotoelettrico**, in questo caso, tutta l'energia del fotone viene trasferita all'elettrone, che viene espulso di conseguenza. Il fotoelettrone acquista un'energia cinetica E_{pe} uguale alla differenza tra l'energia del fotone incidente (E_o) e la sua energia di legame (E_b) [25].

$$E_{pe} = E_o - E_b$$

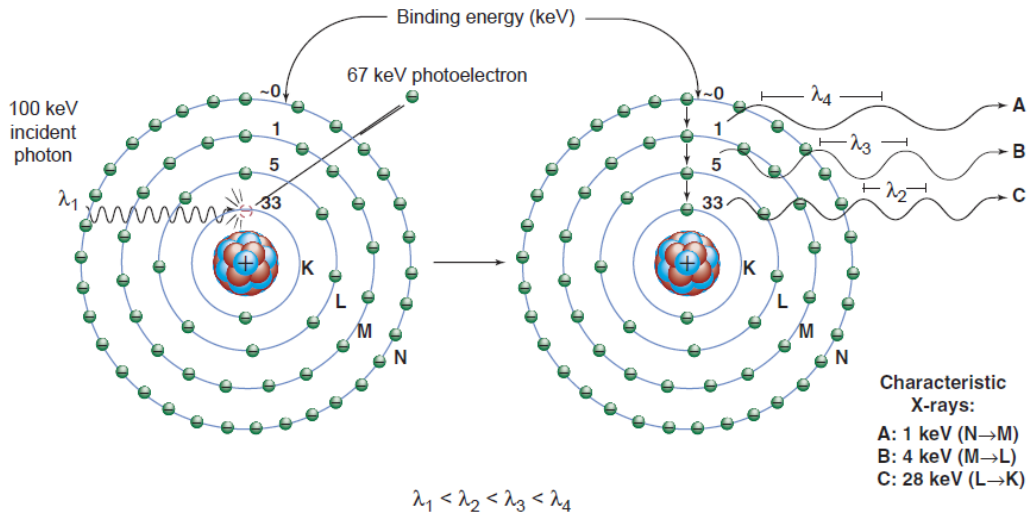


Figura 3.3 Effetto fotoelettrico. ***A sinistra****: un fotone colpisce un elettrone nello strato K, causandone l'espulsione con un'energia cinetica di 67 keV, uguale alla differenza fra l'energia del fotone (100 keV) e l'energia di legame (33 keV).* ***A destra****: il vuoto dovuto all'espulsione dell'elettrone, genera un effetto a catena fino a raggiungere il guscio esterno. Inoltre, ad ogni strato, l'energia viene rilasciata come raggi-x.*

Come mostrato nella figura 3.3, l'espulsione dell'elettrone causa un vuoto nello strato di appartenenza, e di conseguenza, viene colmato da un elettrone di uno stato più esterno. La differenza tra l'energia di legame dei strati, è rilasciata sotto forma di raggi-x. Questo, inoltre, crea un effetto a cascata fino a raggiungere il guscio esterno di elettroni.

L'effetto fotoelettrico si verifica se l'energia del fotone è di poco superiore all'energia di legame, la probabilità scende se l'energia continua ad aumentare, ed è nulla se inferiore.

I meccanismi di interazione, presentati qui sopra, causano il fenomeno dell'attenuazione. A basse energie (meno di 26 keV), l'effetto dominante è quello fotoelettrico, mentre l'effetto di Compton domina ad energie più elevate. Lo Scattering di Rayleigh, come detto precedentemente, ha una bassa probabilità di verificarsi (5-10%). Il numero (n) di fotoni rimanenti dopo che il raggio ha attraversato un materiale di spessore Δx, è calcolato nel modo seguente:

$$n = \mu N \Delta x$$

Dove μ è il coefficiente di attenuazione lineare, ovvero la frazione di fotoni che viene rimossa dal raggio per unità di spessore del materiale, ed è dato da:

$$\mu = \mu_{Rayleigh} + \mu_{effetto\ fotoelettrico} + \mu_{effetto\ Compton}$$

3.2.2 Acquisizione

I componenti principali per l'acquisizione delle immagini sono: il "gantry", una struttura circolare a forma di ciambella, che ruota attorno al paziente per collezionare i dati necessari a costruire il volume; il lettino integrato al sistema, si deve muovere in modo opportuno durante l'esame. All'interno del grantry (figura 3.4) vi è il tubo di scansione che emana i raggi-x e lungo la corona circolare sono posizionati i rilevatori (detettori) in modo stazionario.

Figura 3.4 Scanner per la TAC, composto dal gantry e da un lettino.

Nei modelli di prima generazione, la metodologia di acquisizione si diceva assiale (o sequenziale), i raggi di scansione erano paralleli fra loro, ed ognuno scansionava una piccola fetta. Questo richiedeva un movimento trasversale (far scorrere il paziente lungo l'apertura del gantry) che era altamente inefficiente, in quanto ad ogni movimento del lettino bisognava spegnere il raggio [26].

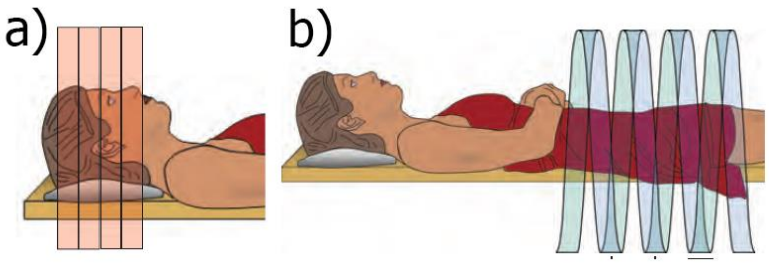


Figura 3.5 Il gantry a scansione assiale (a) produce raggi paralleli, mentre quello a spirale (b) forma un'elica, il che velocizza l'operazione perché in continuo movimento.

L'approccio moderno consiste in un movimento costante e continuo, si forma così una spirale. Non dovendosi più fermare ad ogni scansione, si ottiene che la velocità di acquisizione, rispetto al metodo assiale, è di gran lunga superiore.

3.2.3 Ricostruzione

L'obiettivo di questa fase è quello di costruire un array tridimensionale con valori della scala di grigio. Si può immaginare questa struttura come un parallelepipedo formato da piccoli cubi, detti voxel (volumetric picture element). Durante la scansione non si ottiene il valore dei singoli voxel, ma la loro somma lungo una direzione e, facendo abbastanza scansioni su angoli diversi, è possibile trovare una soluzione unica alla costruzione del volume.

I dati ottenuti dalla scansione indicano il livello di attenuazione del raggio e, trattandosi di un'unità esponenziale, per semplificare viene normalizzata mediante una trasformazione logaritmica, convertendo un problema di tipo esponenziale ad uno lineare. Uno degli algoritmi usati per ricostruire il volume è la *Retroproiezione* [18], del quale viene fornito un esempio di seguito.

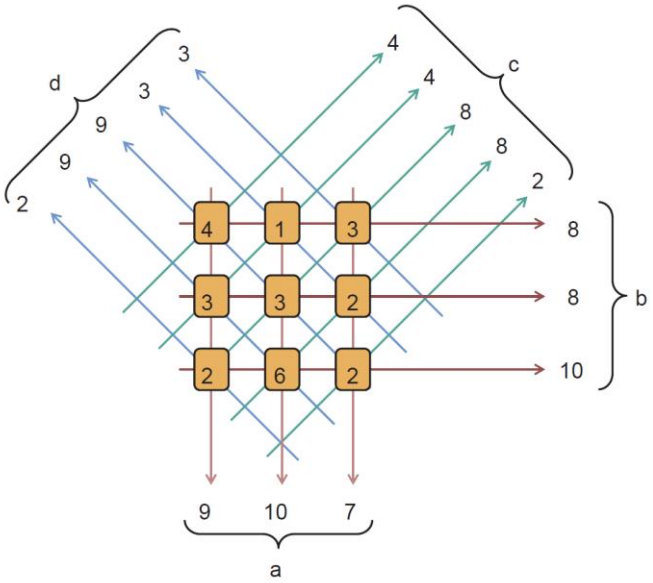


Figura 3.6 La griglia rappresenta i voxel del volume, mentre a-d sono le proiezioni ottenute dalla scansione. I valori dei voxel devono soddisfare tutte le somme nelle diverse direzioni.

La figura 3.6 mostra un esempio semplificato di una sezione di un volume, di dimensione 3x3 voxels (nei casi reali è di circa 512x512), le proiezioni *a-d* sono i dati ottenuti dalla scansione. Si definisce come matrice *M* la griglia in figura 3.6, e i suoi valori come m_{ij}; è possibile rappresentare ogni singolo raggio delle proiezioni *a-d* come un'equazione. Per esempio, l'equazione per il primo raggio della proiezione *a* sarà:

$$m_{11} + m_{21} + m_{31} = 9$$

Ottenendo in questo modo anche le equazioni per i raggi rimanenti, si è in grado di costruire un sistema lineare che, una volta risolto, fornisce i valori della matrice M, i quali, dopo un'opportuna mappatura, possono essere usati direttamente per generare l'immagine. Inoltre, esistono altri algoritmi (più complessi), per la ricostruzione delle immagini TC, come la retroproiezione filtrata e la ricostruzione basata sulle trasformate di *Fourier* [18].

3.3 Risonanza magnetica

Di seguito verranno presentati gli argomenti riguardanti la risonanza magnetica, dai principi fisici come il magnetismo e il fenomeno della risonanza, ai metodi di acquisizione e ricostruzione dell'immagine.

3.3.1 Magnetismo

Il magnetismo è una proprietà fondamentale della materia, generata dal movimento degli elettroni, può essere casuale oppure allineato a un campo magnetico. Quest'ultimo è caratterizzato da un dipolo, ovvero un polo nord e uno sud di carica opposta, e sono generate facendo passare una carica elettrica in una bobina (di rame). Nella RM, un forte campo magnetico genera immagini migliori (più contrasto), i moderni sistemi sono in grado di creare un campo magnetico di 3T (tesla) di potenza [20], che equivale a circa 60'000 volte quello terrestre (0,00005T).

Le particelle subatomiche del nucleo atomico, protoni e neutroni, hanno proprietà magnetiche che dipendono dalla rotazione e dalla carica intrinseca della particella. Entrambe, mediante la rotazione, creano un campo magnetico, riferito come momento magnetico nucleare, ed è rappresentato come un vettore che indica magnitudo e direzione. Quando il numero di protoni e neutroni sono entrambi pari, il momento magnetico è uguale a zero. Mentre, se il numero di uno dei due è dispari, il momento risultante è non-nullo. Infine, disponendo di un'elevata quantità di questi nuclei (circa 10^{15}) in modo non casuale, è possibile osservare il momento magnetico, dal quale derivano i segnali della RM [27].

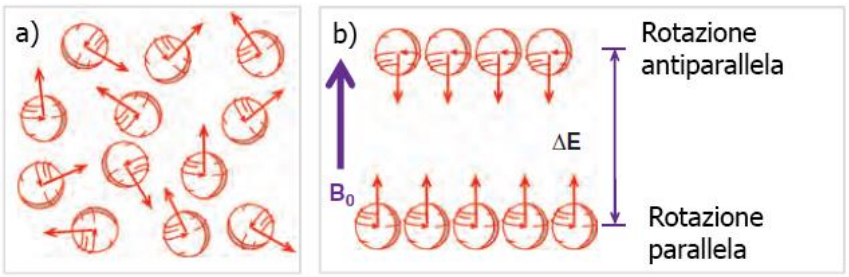


Figura 3.7 A) Protoni liberi, senza campo magnetico. B) Protoni allineati da un campo magnetico.

La RM usa i protoni dell'idrogeno che, in quanto contenute nelle molecole d'acqua, sono abbondanti nei tessuti. In presenza di un campo magnetico potente, i protoni dell'idrogeno, da un una disposizione casuale, si allineano in due modi: parallelo e antiparallelo al campo magnetico (figura 3.7). Inoltre, i protoni sono soggetti alla precessione giroscopica, che consiste in una rotazione intorno al proprio asse simile a quello di una trottola. L'equazione di *Larmor* [18], determina la frequenza di precessione (ω_0):

$$\omega_0 = \gamma * B_0$$

dove γ indica il rapporto giromagnetico unico per ogni elemento (γ_H = 42.58 MHz/T), mentre B_0 è la potenza del campo magnetico. La precisione della frequenza di precessione è essenziale, poiché deve combaciare con la radiofrequenza, affinché l'energia venga assorbita dai protoni.

3.3.2 Risonanza

Per scoprire le caratteristiche di un sistema fisico, è necessario perturbare il sistema dallo stato d'equilibrio per poi misurare l'energia emessa ritornando all'equilibrio. Sincronizzando l'emissione delle radiofrequenze alla frequenza di precessione del protone causa l'assorbimento dell'energia e la perturbazione dell'equilibrio del momento magnetico. Il rilascio dell'energia avviene quando il protone ritorna allo stato d'equilibrio, causando il fenomeno della risonanza [27].

L'impulso della radiofrequenza è riprodotto per un tempo molto breve, nell'ordine di millisecondi, quanto basta per ribaltare il protone di 90° [18]. Il segnale misurabile che viene prodotto dalla fase di rilassamento, sotto forma di corrente, è detto FID (*free induction decay*). Ci sono due tipi rilassamenti:

- Trasversale **T2**, è il tempo necessario affinché l'energia riemessa dal protone decada al 37% del valore di picco (subito dopo l'impulso RF di 90°). Rappresenta la desincronizzazione dei protoni dai rispettivi vicini.

- Longitudinale **T1**, viene misurato il tempo per ritornare allo stato d'equilibrio, ovvero quando ogni protone rilascia completamente l'energia e ritorna allineato al campo magnetico. Rispetto a T2, richiede un tempo maggiore.

3.3.3 Acquisizione e ricostruzione

L'output della risonanza magnetica fornisce un segnale prodotto da tutto il tessuto nello scanner, questo vuol dire che, senza un modo per localizzare i segnali, si otterrebbe solamente un numero per l'intero zona. Lo scopo dell'acquisizione è quello di trovare le coordinate z, x e y dei voxel che compongono il volume.

La prima dimensione (z) è lo slice (o fetta), è localizzata mediante l'uso del gradiente [28] che è un campo magnetico non costante, ovvero che cambia intensità a seconda della posizione, i scanner RM producono 3 gradienti (lungo z, x e y). Grazie al gradiente, è possibile selezionare una qualsiasi slice e, in quanto la risonanza avviene solamente se la frequenza di Larmor combacia con l'impulso RF, basterà modificare la RF in modo opportuno. Una volta che si ha trovato una slice, bisogna localizzare lungo gli assi x e y. Il gradiente, *Frequency encode gradient* (FEG), è applicata nella direzione perpendicolare al gradiente della slice, lungo l'asso x. In questo modo la frequenza di precessione dei protoni della slice sarà diversa, perciò viene usata la trasformata di *Fourier* per riuscire a separare il segnale in base alle diverse frequenze. Infine, per localizzare l'ultima coordinata, invece di cambiare la frequenza, è necessario manipolare la *fase* dei protoni. La fase indica la variazione dal punto di partenza della frequenza di precessione, ovvero questa viene traslata rispetto alla frequenza originale. La manipolazione avviene, tramite la *Phase Encode Gradient* (PEG), accendendo e spegnendo il gradiente lungo l'asse y. La posizione è quindi determinata dall'ammontare del cambio di fase; per esempio, i protoni al centro del raggio d'azione avranno un cambiamento di fase nullo, mentre, avvicinandosi ai lati, i protoni avranno uno slittamento di fase sempre maggiore.

I dati della RM sono memorizzati in una matrice k-spazio (figura 3.8), che è una matrice bidimensionale di valori dei segnali acquisiti [18]. La matrice è suddivisa in quattro quadranti, dove l'origine è posta al centro con frequenza uguale a 0. Lungo l'asse delle ascisse ci sono i valori ottenuti dalla FEG, e sull'asse delle ordinate quelli della PEG.

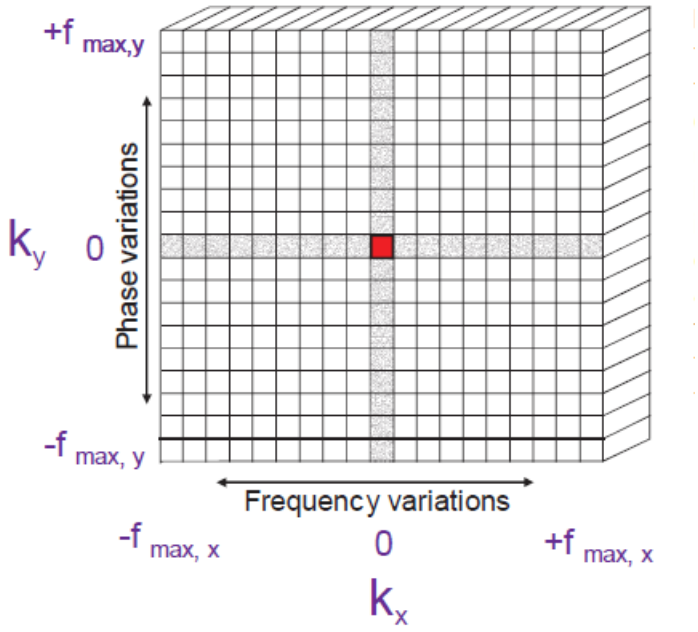


Figura 3.8 Matrice k-spazio, su di essa vengono salvati i valori dei segnali acquisiti.

Una volta che la matrice k-spazio è riempita, viene usato un inverso della trasformata di Fourier per decodificare le variazioni di fase di ogni colonna per produrre l'immagine. L'immagine finale è mappata sui livelli di grigio, dove ogni pixel corrisponde a un voxel.

3.4 Formato dei file

Nel campo della diagnostica per immagini, l'output viene salvato su file in formati ad hoc, questo perché le informazioni sono molteplici e devono essere organizzate nel modo corretto; per esempio, nel caso di un volume, è fondamentale sapere l'ordine numerico dell'immagine per determinare la zona a cui appartiene. Perciò, il formato del file descrive come sono organizzate le immagini e come interpretare i dati per una visualizzazione corretta.

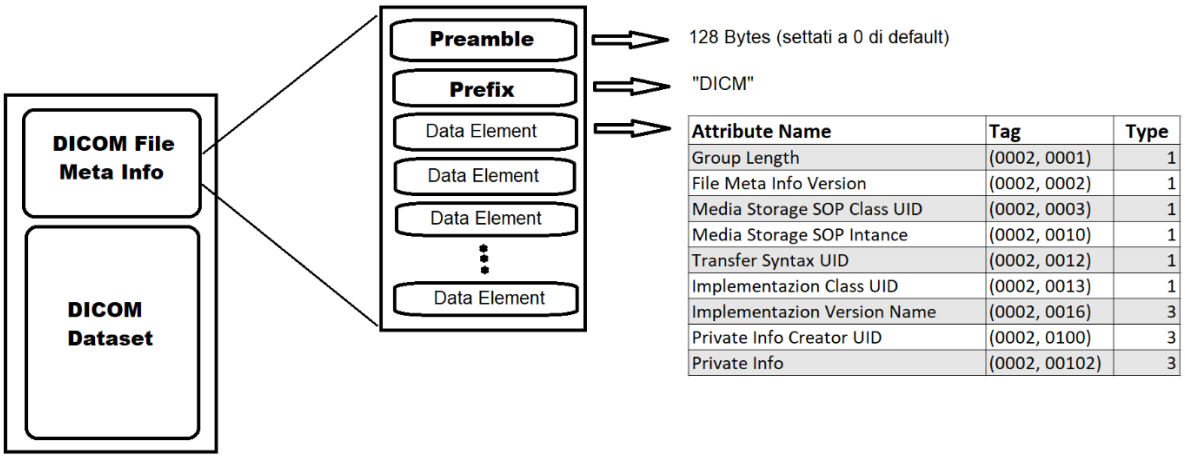
Le caratteristiche comuni a tutti i tipi di formato sono [29]:

- ***Pixel depth***, è la profondità di colore, rappresenta il numero di bit usati per codificare i singoli pixel, cioè il numero di sfumature del colore. Siccome l'unità minima

memorizzabile sul computer è il byte, sia per un *pixel depth* di 10 bit che uno di 16 bit, verranno usato 2 bytes per pixel.

- ***Pixel Data***, sono i valori dei pixel effettivamente salvati. Questi valori possono essere numeri interi, *floating point* o anche complessi. Tipicamente le immagini RM e TAC usano degli interi di tipo *unsigned* a 16 bit. I valori *floating point* e i numeri complessi sono, invece, usati quando si ha la necessità di salvare più informazioni; ad esempio, i numeri complessi possono essere usati per salvare la matrice k-spazio di una RM, prima della sua ricostruzione.
- ***Photometric Interpretation***, indica come interpretare i *pixel data*, ovvero come immagini monocromatiche o a colori. Per specificare il colore vengono adoperati i canali, ad esempio RGB per immagini colorate.
- ***Metadata***, rappresentano le informazioni che descrivono l'immagine. Comunemente salvata all'inizio del file, include le dimensioni delle immagini, la risoluzione, *photometric interpretation* e *pixel depth.* Grazie ai metadati è possibile aprire e usare correttamente il file. Inoltre, possono essere salvati anche informazioni aggiuntive come la modalità di scansione e i parametri utilizzati.

Il formato più diffuso è, senza dubbio, la **Dicom**, nato nel 1993 dalla ACR (American College of Radiology) e dalla NEMA (National Electric Manufactures Association). Questo standard introduce il concetto di dipendenza dei pixel data dai metadati, ovvero che le immagini diventano insignificanti se separate dai metadati.



Attribute Name	Tag	Type
Group Length	(0002, 0001)	1
File Meta Info Version	(0002, 0002)	1
Media Storage SOP Class UID	(0002, 0003)	1
Media Storage SOP Intance	(0002, 0010)	1
Transfer Syntax UID	(0002, 0012)	1
Implementazion Class UID	(0002, 0013)	1
Implementazion Version Name	(0002, 0016)	3
Private Info Creator UID	(0002, 0100)	3
Private Info	(0002, 00102)	3

Figura 3.9 Struttura del formato DICOM. L'header contiene i campi per i metadati.

La struttura del file (figura 3.9) è costituita da due elementi: *header*, *data set*. L'header è composto da una stringa di 128 bytes, chiamata *Preamble*, seguito dal *prefix*, un blocco di 4 byte che contiene i caratteri "DICM". I metadati sono codificati nell'header in appositi campi (data element) e non hanno una dimensione predefinita. Il blocco dei *data set* contiene le immagini della scansione, che possono essere multiple. Inoltre, Dicom supporta solamente i tipi *integer* per i pixel data [29].

Il formato **Nifti** fu creato nel 2003 dalla National Institute of Health per sostituire, l'allora popolare, formato Analyze 7.5 che era carente di informazioni riguardanti l'orientamento nello spazio. Nifti, come per Dicom, è caratterizzato da un solo file ".nii" che include sia l'header che i pixel data, ma per motivi di compatibilità è possibile anche salvarli separatamente su due file con estensione *.hdr/.img*. L'header questa volta ha una dimensione fissa di 348 bytes, e contiene informazioni riguardanti le modalità di acquisizione e la struttura delle immagini. Alcuni campi dell'header sono riportati nella tabella di seguito [30]:

Tabella 3.2 **Le principali voci dell'header nel formato Nifti**

Tipo	Nome	Offset	Size	Descrizione
int	sizeof_hdr	0B	4B	Indica la dimensione dell'header, che deve 348.
short	dim[8]	40B	16B	Specifica la dimensione dell'image array.
int	datatype	70B	2B	Specifica la *photometric intepretation.*
short	slice_start	74B	2B	Indica la prima slice acquisita.
float	pixdim[8]	76B	32B	Specifica la dimensione dei voxel.
short	slice_end	120B	2B	Indica l'ultima slice acquisita.
char	slice_code	122B	1B	Codice che identifica il tipo di scansione.
float	slice_duration	132B	4B	Indica il tempo per la scansione di un slice.
char	descrip[80]	148B	80B	Una arbitraria stringa di 80 caratteri.
char	intent_name[16]	328B	16B	Nome o significato dei dati.
char	magic[4]	344B	4B	Stringa magica. Serve per indicare la versione.
Dimensione Totale = 348B				

Il formato Nifti è riuscito a diventare lo standard nel campo della neuroscienza, grazie a un ampio supporto da parte dei software di visualizzazione. E nel 2011, è stata rilasciata una nuova versione, la Nifti-2, la quale permette di rappresentare le dimensioni delle matrici immagine a 64 bit invece dei 16 bit originali, rendendo possibile codificare immagini più grandi. Inoltre, è stato ulteriormente allargato l'header, passando da 348 a 544 bytes.

Capitolo 4

Rendering volumetrico

Questo capitolo sarà dedicato all'argomento principale di questo elaborato, ovvero il rendering volumetrico di immagini mediche. Rispetto al rendering tradizionale (poligonale), il volume rendering, ricrea l'intero oggetto, non sola la superficie esterna. In questo modo, non si hanno perdite di informazioni ed è possibile osservare dettagli nascosti all'interno, caratteristica che la rende particolarmente adatta alla diagnostica per immagini.

4.1 Visualizzazione di dati volumetrici

Come menzionato nel capitolo precedente, i dati delle scansioni TAC e RM sono costituiti da un insieme di immagini sovrapposte, simile a una pila di fotografie, e si presentano sotto forma di un array tridimensionale di valori scalari. Il volume rendering permette di visualizzare le scansioni tomografiche come oggetti tridimensionali, il che rende più semplice la localizzazione dei punti d'interesse e, inoltre, fornisce anche una rappresentazione della densità dei tessuti.

Il dataset è composto da un insieme *V* di valori (*x*,*y*,*z*,*v*), chiamati *voxel,* termine che deriva da *volume* e *element,* e rappresenta la controparte per i volumi dei *pixel* delle immagini 2D. Sono caratterizzati da una posizione nello spazio 3D (*x*,*y*,*z*) e da un valore *v* che può essere binario, se assume solo valori 0 o 1, multi-valore, se indica una cerca proprietà. È possibile definire i valori mediante la una funzione:

$$f(\vec{v}) \in R \; con \; \vec{v} \in R^3$$

In quanto i dati forniti sono discretizzati, per rappresentare completamente i dati originali, è necessario eseguire una ricostruzione per ottenere la funzione continua *f.*

4.1.1 Il processo di visualizzazione

Per ottenere la visualizzazione del volume bisogna seguire una serie di step, la figura 4.1 mostra quali sono le azioni da effettuare ad ogni passo del rendering.

Il procedimento può essere riassunto nelle seguenti fasi:

1. **Acquisizione** dei dati di input. Il quale può contenere fase preparatoria dei dati, in modo da migliorare la qualità finale.
2. **Classificazione** del volume, consiste nell'assegnazione del colore e dell'opacità ai voxel.
3. **Shading**, determina il modello di illuminazione. Aggiunge la percezione di profondità, facilitando la visualizzazione grazie al senso di tridimensionalità.
4. **Compositing**, proiezione su schermo delle informazioni ottenute.

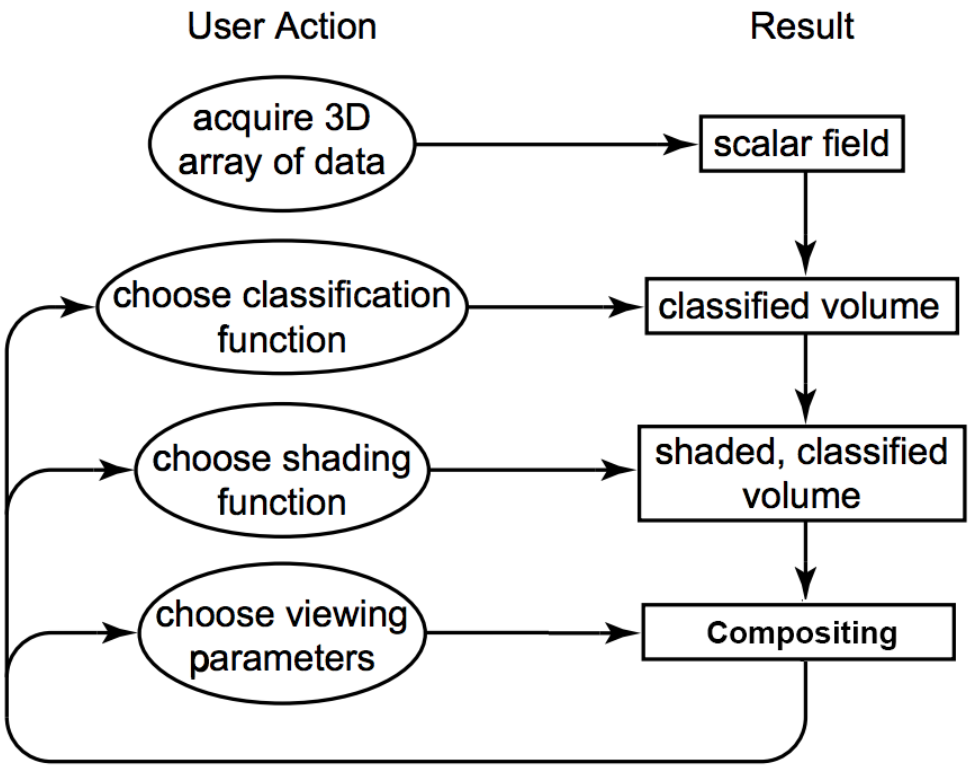


Figura 4.1 Il processo di visualizzazione contiene le tre fasi principali del rendering volumetrico: classificazione, shading e compositing. Fonte [31].

4.1.2 Campionamento e ricostruzione

I dati del volume sono ottenuti tramite il campionamento del segnale continuo originale, questo vuol dire che è avvenuta una discretizzazione del segnale (figura 4.2). Per visualizzare il volume campionato, il segnale deve essere ricostruito alla forma originale, ovvero continua. Secondo la teoria del campionamento, è possibile ricostruire perfettamente il segnale di partenza, se questa ha una banda limitata e se il campionamento è stato effettuato ad almeno il doppio della frequenza massima, detta anche frequenza di *Nyquist* [32]. La banda limitata significa che non ci devono essere frequenze al di sopra di una certa soglia.

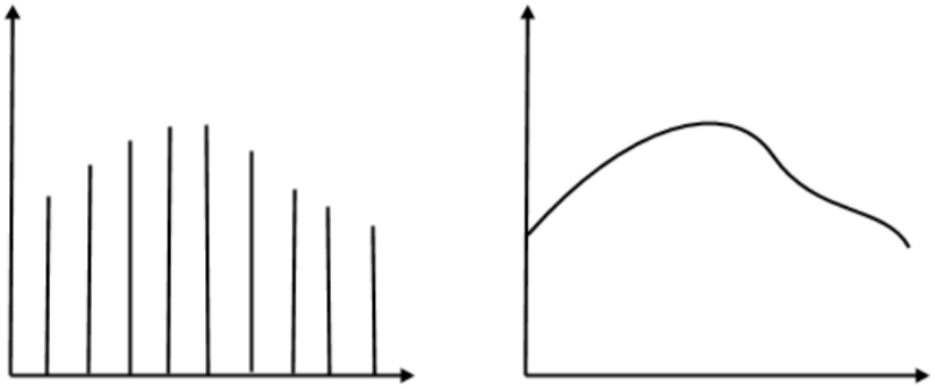


Figura 4.2 Segnale campionato (sinistra). Segnale continuo originale (destra).

Sfortunatamente, i dati prodotti contengono quasi sempre infinite frequenze [42], e perciò, non è possibile effettuare una ricostruzione perfetta. Questo causa, nell'immagine finale, l'effetto di aliasing, ovvero un'immagine seghettata e poco pulita.

Per ridurre l'aliasing vengono usati dei filtri. Il filtro ideale (per la ricostruzione perfetta) è il filtro *sinc*, ma avendo estensione da -∞ a +∞, non può essere implementato in pratica, perciò vengono comunemente usati filtri come il *box filter* e il *tent filter*. Questi filtri agiscono, rimuovendo tutte le frequenze al di sopra di un limite predefinito. Il *tent filter*, detto anche filtro lineare, applica una interpolazione trilineare dei campioni vicini. Inoltre, esistono filtri più complessi come il filtro b-spline e il *Blackman windowed sinc* [34] che producono immagini migliori, ma al costo di tempi di calcolo maggiori.

4.2 Tecniche di volume rendering

Esistono molti metodi per ottenere la visualizzazione di valori scalari nello spazio tridimensionale, questi possono essere classificati nelle seguenti tecniche generali.

4.2.1 Volume rendering indiretto

Il volume rendering indiretto consiste nel convertire i valori dei dati volumetrici in una rappresentazione intermedia, fatta di primitive geometriche, per poi essere renderizzata con tecniche poligonali. I valori sono scelti da una iso-superficie che è, per definizione, la superficie in un campo continuo *F(x,y,z)*, nel quale il valore è uguale a un iso-valore predefinito [35]. Metodi per il volume rendering indiretto sono: *surface tracking*, *iso-surfacing*, e *domain-based* rendering.

Vantaggi di questa tipologia sono costituiti dalla velocità e flessibilità del rendering; ad esempio, nella pianificazione di un intervento maxillofacciale, le superfici poligonali possono essere manipolate e riposizionate in modo interattivo [36]. La conversione comporta degli svantaggi, in quanto una parte dei dati viene perduta, infatti, in media le superfici sono derivate solamente dal 10% dei dati disponibili [36], di conseguenza, potrebbero emergere casi di falsi positivi (superfici inesistenti) o falsi negativi (buchi all'interno di superfici rilevate) [37], oltre a non mostrare la variazione di densità dei tessuti.

4.2.2 Volume rendering diretto

Le tecniche di volume rendering diretto producono l'immagine usando l'intero dataset, senza estrarre esplicitamente una superficie basata su iso-valori. Il rendering avviene su tutti i voxel del volume, e include un modello di illuminazione che supporta voxel semitrasparenti; in modo che ognuno di essi siano potenzialmente visibili. Si devono, perciò, specificare le proprietà fisiche che andranno ad influire la rappresentazione del volume, quali, l'emissione, l'assorbimento, riflessione e rifrazione. I valori del dataset volumetrico vengono, quindi, mappati in colore e opacità durante il rendering. Questo mapping costituisce la fase di classificazione, e il risultato è determinato dalla *transfer function* usata.

Il modo più semplice è quello di proiettare i voxel sul Z-buffer come punti 3D, per poi disegnarli a schermo (*compositing*). I voxel vengono considerati con una spaziatura uniforme in tutte le direzioni. Se due voxel vengono proiettati sullo stesso pixel del piano di proiezione, prevale quello che arriva dopo. Questo problema può essere risolto mediante gli algoritmi:

- **Back-to-front (BTF)**: i voxel vengono visitati nell'ordine inverso, dal più lontano a quello più vicino al punto di osservazione. In questo modo si sovrascrivono i voxel più lontani. Metodi di questo tipo sono chiamati anche "algoritmo del pittore" o "*list-priority algorithms*". BTF funziona correttamente per le proiezioni ortografiche, ma non produce risultati corretti per quelle prospettiche [37].
- **Front-to-back (FTB)**: essenzialmente lo stesso dell'algoritmo BTF, con la differenza che i voxel sono attraversati a partire dall'osservatore, verso il punto più lontano. Quindi è possibile stabilire quali voxel influiscono sulla visualizzazione finale.

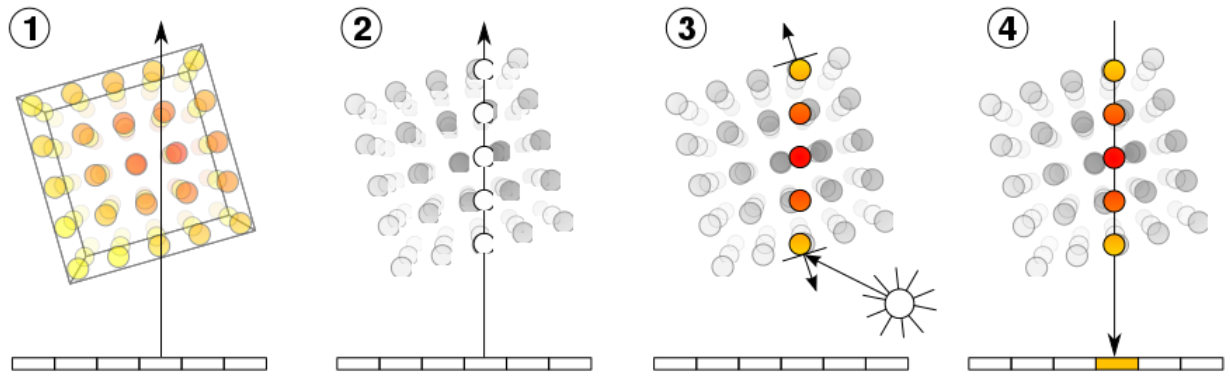


Figura 4.3 Le fasi del metodo ray casting: Ray casting -> Sampling -> Shading -> Compositing

Un metodo più complesso è il volume ray casting (figura 4.3), per ogni pixel dell'immagine finale viene generato un raggio. Il raggio parte dalla camera (punto di osservazione) e passa attraverso un piano di proiezione, per poi colpire il volume. Lungo il tragitto, il volume viene campionato, ma siccome i valori del volume presentano spazi, è necessario applicare un filtro di interpolazione (comunemente si usa l'interpolazione trilineare). Successivamente, per ogni punto campionato, viene applicato uno shader, e infine, viene calcolato il colore finale da assegnare al pixel.

4.2.3 Texture based

L'approccio texture based fu la prima tecnica di rendering volumetrico basata su GPU, fu ideato da Cullip e Newman [38]. Questo metodo immagazzina i dati volumetrici direttamente nella memoria della GPU, sotto forma di una pila allineata di texture 2D. Le texture vengono mappate su dei proxy, una sequenza di slice 2D semitrasparenti. I proxy hanno come funzione quella mostrare le texture, e il rendering avviene usando un approccio back-to-front, sfruttando l'accelerazione hardware *alpha blending*, offerta dalle GPU.

L'uso delle texture 2D, rende necessario tre copie del dataset volumetrico, ognuno allineato su uno degli assi, incrementando, di conseguenza, la memoria necessaria di tre volte. La visualizzazione prevede che solo una di queste pile o *stack* venga effettivamente mostrata; viene mostrata quella che si trova maggiormente parallela al vettore di visualizzazione. Quando una texture viene mappata sul proxy, l'informazione delle slice precedenti e di quelle successive non viene tenuta in considerazione. Risultando in artefatti visibili e, generalmente, una bassa qualità visiva.

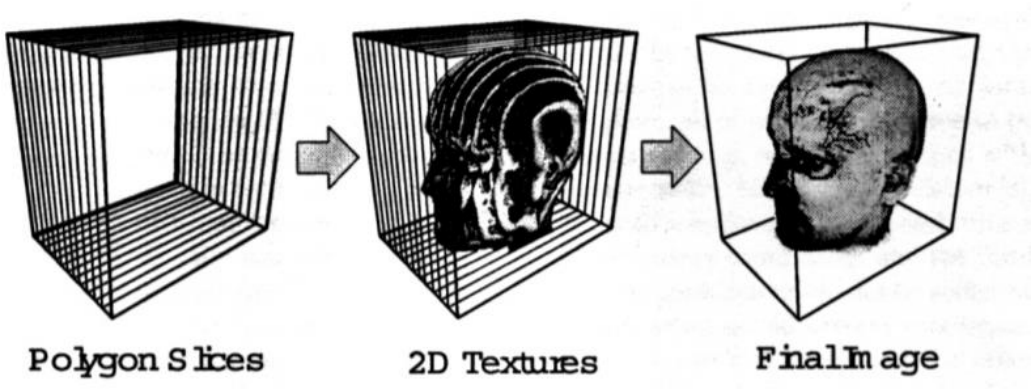


Figura 4.4 Metodo texture based mediante texture 2D. Fonte [39]

Fortunatamente, si possono usare texture 3D, che risolvono parte dei problemi; per esempio, con le texture 3D, sola una copia dei dati è richiesta, riducendo i requisiti di memoria. Inoltre, grazie all'interpolazione trilineare, che a differenza di quella bilineare usata dalle texture 2D, è possibile estrarre slices da qualsiasi direzione; risultando in una qualità dell'immagine superiore.

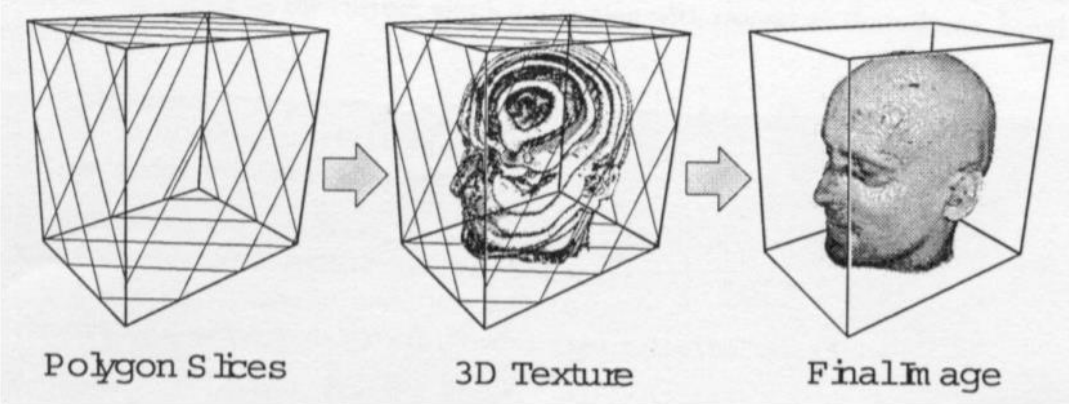


Figura 4.5 Metodo texture based mediante texture 3D. Fonte [39]

4.2.4 Maximum Intensity Projection

Si tratta di una variante del volume rendering diretto, il Maximum Intensity Projection determina il colore finale di ogni pixel in base al valore massimo incontrato lungo il percorso del raggio di ray casting. Questa tecnica è particolarmente adatta alle immagini prodotte dalla risonanza magnetica che, tipicamente, soffrono di un alto tasso di rumore, e quindi, risulta difficile interpretare i dati. Ad esempio, i valori delle strutture vascolari, acquisite mediante scansione RM, sono più alti di quello dei tessuti circostanti. Sfruttando questo principio si possono ottenere immagini molto più chiare, rispetto al volume rendering diretto, come mostrato in figura 4.7.

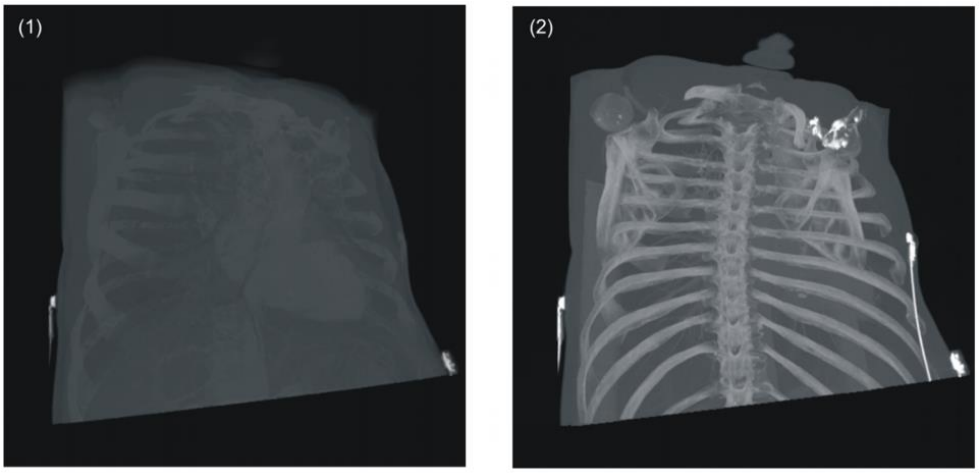


Figura 4.6 Confronto tra volume rendering diretto (1) e Maximum Intensity Projection (2)

4.3 L’equazione del rendering volumetrico

Come per il rendering tradizionale, è necessario un modello fisico che definisce come i voxel interagiscono con la luce. A seconda del livello di dettaglio che si vuole ottenere, il modello può includere proprietà fisiche, quali l’emissione, assorbimento, riflessione e diffusione della luce. Il modello più utilizzato è quello di Blinn [40]. In quanto la particolarità del volume rendering è l’assenza di superfici, Blinn ha considerato i valori di densità dei dati volumetrici come particelle sferiche, i quali interagiscono con la luce in base alla densità. Gli effetti che vengono considerati sono:

- **Scattering (diffusione)**: descrive la deviazione della direzione e frequenza dei fasci di radiazioni dovuto all’interazione con la materia.
- **Assorbimento**: processo dove l’energia della luce viene trasformato in calore.
- **Emissione**: quando la luce viene introdotta nella scena da una sorgente esterna.

Data la complessità computazionale del calcolo dello scattering, questa non è incluso nel modello di Blinn, e quindi si tratta di un modello *emissione-assorbimento.* La parte dell’emissione determina l’intensità e colore della particella all’interno del volume, mentre, l’assorbimento determina l’opacità.

4.3.1 L’equazione dell’assorbimento

In un sistema con solo assorbimento, senza emissione e scattering, vale la seguente equazione differenziale [41]:

$$\frac{dI}{ds} = -\tau(s)I(s)$$

dove *s* indica la distanza del raggio, *I(s)* l’intensità della luce in relazione a *s*, infine *τ(s)* è il coefficiente di opacità. Risolvendo l’equazione si ottiene [41]:

$$I(s) = I_0 e^{-\int_0^s \tau(t)dt}$$

dove I_0 è l’intensità iniziale del raggio.

4.3.2 L’equazione dell’emissione

Considerando solamente l’emissione, si avrà l’equazione differenziale:

$$\frac{dI}{ds} = c(s)\tau(s)$$

dove *c(s)* è il coefficiente di emissione. La soluzione all’equazione sarà [45]:

$$I(s) = I_0 + \int_0^s c(t)\tau(t)dt$$

4.3.3 L’equazione finale

L’equazione del volume rendering è formata dalla combinazione dell’assorbimento e dell’emissione, di cui si propone la formula di seguito [45]:

$$\frac{dI}{ds} = c(s)\tau(s) - \tau(s)I(s)$$

dalla quale si ricava la seguente soluzione [45]:

$$I(D) = I_0 e^{-\int_0^D \tau(t)dt} + \int_0^D c(t)\tau(t)\, e^{-\int_s^D \tau(t)dt} ds$$

dove D è la posizione dell’osservatore. Si nota che l’equazione è formata da due parti, la prima è la formula dell’assorbimento e descrive l’assorbimento della luce ambientale, mentre, la seconda descrive come il volume emette e assorbe la luce in arrivo.

4.3.4 L'equazione discretizzata

Data la difficoltà dei metodi analitici per i computer, nella pratica si usa un metodo numerico che comporta un'approssimazione. La versione discretizzata è ottenuta mediante l'applicazione della somma di Riemann, dove il raggio di visualizzazione è suddiviso in segmenti uniformi. Di conseguenza, la parte dell'assorbimento può essere riscritta nel modo seguente [45]:

$$e^{-\int_0^D \tau(s)dt} \approx e^{-\sum_{i=1}^{n} \tau(i\Delta x)\Delta x} \approx \prod_{i=1}^{n} e(-\tau(i\Delta x)\Delta x) \approx \prod_{i=1}^{n} t_i$$

dove $t_i = (-\tau(i\Delta x)\Delta x)$ può essere considerato come la trasparenza dell'i-esimo segmento lungo il raggio. Inoltre, considerando $c(s)\tau(s) = g(s)$, la somma di Riemann della seconda parte dell'equazione diventa [45]:

$$\int_0^D g(s)\, e^{-\int_s^D \tau(t)dt} ds \approx \sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_j$$

Quindi l'equazione discretizzata diventa:

$$I(D) = I_0 \prod_{i=1}^{n} t_i + \sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_j$$

4.4 Classificazione dei voxel

Una delle operazioni più importanti durante il rendering volumetrico è quello della classificazione. Si tratta del processo che ha il compito di mappare i valori del dataset in colore (c) e in opacità (τ), tramite le *transfer function*. L'obiettivo della classificazione è quello produrre un'immagine in grado di mettere in risalto le caratteristiche del volume; ad esempio, mostrare un contrasto adeguato, che permetta di distinguere i vari tipi di tessuti.

Le tipologie di classificazione si suddividono in:

- **Pre-classificazione**, quando lo step di classificazione avviene prima dello step di ricostruzione della funzione continua, nella pipeline di rendering. La pre-classificazione applica una interpolazione sui valori dei colori e dell'opacità. Questo comporta errori nella ricostruzione, producendo artefatti visibili; ma ha il vantaggio di essere molto più performante, in quanto è eseguita nella fase di pre-processing.
- **Post-classificazione**, il processo di classificazione è applicato sulla funzione continua del segnale, invece di quello discreto. Questo genera immagini di qualità superiore rispetto alla pre-classificazione, ma al costo di un overhead computazionale.

4.4.1 Transfer functions

Le transfer function sono usate dall'utente e servono per mappare i valori volumetrici in colore e opacità. Perciò, esse devono essere in grado di mostrare l'area del volume d'interesse dell'utente e, allo stesso tempo, essere abbastanza semplici in modo che l'utente possa capirne i meccanismi.

Tipicamente viene fatto uso della transfer function monodimensionale, in quanto è quella più semplice. In questa funzione, solamente un valore viene usato per il mapping, ovvero il valore di densità dei voxel. L'utente, in questo modo, dovrà solo considera la densità per manipolare l'output. Ovviamente, da questa implementazione sorgono alcuni problemi; per esempio, effetti di aliasing sono molto comuni e inoltre, manipolando solamente il livello di densità, risulta difficile esaltare una particolare area.

Tuttavia, esistono transfer functions multidimensionali che risolvono i problemi sopraccitati. Viene inclusa la derivata prima dei valori del volume, nel caso delle funzioni bidimensionali, anche la derivata seconda, nel caso di funzioni tridimensionali.

4.5 Modelli di illuminazione e gradiente

Attraverso l'illuminazione è possibile creare l'illusione della percezione di profondità in un'immagine bidimensionale, comunicando, potenzialmente, maggiori informazioni all'utente.

4.5.1 Modello di illuminazione di Phong

Il modello di illuminazione più usato in computer graphics, valido anche per il volume rendering, è sicuramente il modello di Phong. Si tratta di un modello di illuminazione locale, ovvero, un punto è illuminato solamente dalla sorgente e non ammette la luce indiretta, in quanto non vengono considerate le riflessioni all'interno dell'ambiente. Il resto della scena è illuminato da una luce ambientale. Il modello di Phong è composto da tre differenti classi di modelli di illuminazione: luce ambientale, riflessione diffusa, riflessione speculare; quindi, l'intensità *I* sarà data da:

$$I = I_{ambientale} + I_{diffusa} + I_{speculare}$$

dove la luce ambientale si ricava dal prodotto fra l'intensità I_a (costante) per il coefficiente di riflessione ambientale:

$$I_{ambientale} = I_a k_a$$

La riflessione diffusa, chiamata anche riflessione Lambertiana, riflette la luce uniformemente in tutte le direzioni, quindi la superficie appare ugualmente luminosa dal qualunque punto di vista. L'equazione è la seguente:

$$I_{diffusa} = I_p k_d (\overline{N} * \overline{L})$$

dove I_p è l'intensità della sorgente, k_d il coefficiente di riflessione diffusiva, $\overline{N}$ e $\overline{L}$ sono i vettori normalizzati del raggio luminoso ($\overline{L}$) e la normale alla superficie nel punto d'incidenza ($\overline{N}$).

Infine, la riflessione speculare produce un raggio riflesso (R), che dipende dalla lucidità della superficie.

$$I_{speculare} = I_p k_s (\bar{R} * \bar{V})^n$$

dove k_s è il coefficiente di riflessione speculare, $\bar{R}$ il vettore di riflessione normalizzato, $\bar{V}$ il vettore normalizzato della direzione dell'osservatore, *n* è una potenza che serve per approssimare il decadimento della riflessione per angoli non allineati con il vettore *V*.

L'equazione completa dell'illuminazione diventa:

$$I = I_a k_a + I_p [k_d (\bar{N} * \bar{L}) + k_s (\bar{R} * \bar{V})^n]$$

4.5.2 Gradiente

I modelli di illuminazione sono dipendenti dalle normali delle superfici, ma le tecniche di volume rendering sono caratterizzate dal fatto di non avere una superficie ben definita, e di conseguenza, non si hanno le normali. Questo problema, è risolto dall'uso del vettore gradiente in tutti punti coinvolti, al posto della normale della superficie. Il vettore gradiente è definito come la derivata prima di un volume discretizzato [41]:

$$df(x,y,z) = (\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz})$$

Siccome il gradiente è definito lungo intervalli infinitesimali, non è possibile calcolarlo direttamente. Tuttavia, esistono diversi metodi per ottenere una stima. Quello più popolare è l'algoritmo *central difference*, il quale usa i vicini sei valori dei voxel nelle direzioni $\pm x, y$ e z per calcolare il gradiente. Può essere scritto nel seguente modo:

$$\begin{aligned} df(x,y,z) = \quad & \frac{1}{2}\big(f(x+1,y,z) - f(x-1,y,z)\big), \\ & \frac{1}{2}\big(f(x,y+1,z) - f(x-1,y-1,z)\big), \\ & \frac{1}{2}\big(f(x,y,z+1) - f(x-1,y,z-1)\big), \end{aligned}$$

L'algoritmo *central difference* è facilmente implementabile e veloce da eseguire, ma può introdurre errori numerici che causano fenomeni di smoothing [43].

4.6 Compositing

Nel volume rendering viene fatto partire un raggio per ogni pixel del piano di proiezione, i raggi lungo il loro percorso determinano il colore e l'opacità. L'obiettivo del compositing è quello di unire tutte le informazioni raccolte dal raggio, allo scopo di produrre il colore del pixel finale. Ci sono due approcci diversi al compositing: *front-to-back* o *back-to-front.*

4.6.1 Front to back

I voxel vengono valutati lungo il raggio a partire dall'osservatore. Il colore e l'opacità finale viene calcolato nel seguente modo [43]:

$$C_{out} = C_{in} + (1 - A_{in})A_i C_i$$

$$A_{out} = A_{in} + A_i(1 - A_{in})$$

dove C_{in} e A_{in} sono la somma del colore e del valore alfa (trasparenza) dei passi precedenti (lungo il raggio), C_i e A_i i valori del passo corrente, e C_{out} e A_{out} sono i nuovi valori ottenuti dalla composizione. Quando A raggiunge il valore 1.0, il contributo dei voxel successivi diventa trascurabile e, perciò, sarà possibile terminare prematuramente il processo di composizione. Questa ottimizzazione è chiamata *early ray termination* [44].

4.6.2 Back to front

La valutazione dei voxel avviene nell'ordine opposto, da quello più lontano a quello più vicino. L'algoritmo diventa [43]:

$$C_{out} = C_i A_i + C_{in}(1 - A_{in})$$

Il vantaggio dell'approccio *back-to-front* consiste nel non dover tenere in considerazione il valore *A* dei passi precedenti. In questo modo, l'esecuzione risulterà leggermente più veloce, ma allo stesso tempo, si perderà la possibilità di applicare *l'early ray termination.*

4.7 Algoritmo di Ray-Casting

In questa parte si discuterà l'implementazione dell'algoritmo di ray casting, sviluppato da Levoy nel 1988 [46].

L'algoritmo consiste nello "sparare" un raggio al volume, per ogni pixel dell'immagine finale. I raggi vengono divisi uniformemente, e lungo il loro percorso, i valori colore e opacità vengono ottenuti tramite interpolazione. Successivamente, questi valori vengono uniti per ricavare il colore finale del pixel (fase di compositing).

La tecnica descritta da Levoy è caratterizzata da due pipeline: una di visualizzazione e una di classificazione. L'output prodotto da queste due pipeline viene poi combinato per produrre l'immagine finale. Nella visualizzazione avviene lo shading, ad ogni voxel è assegnata una ombreggiatura (shade) tramite la normale ottenuta dal gradiente. La normale è, quindi, usata come input al modello di illuminazione di Phong, il quale restituisce come output, l'intensità dei colori per ogni voxel del dataset. Infine, la pipeline di classificazione, assegna un'opacità ad ogni voxel.

La figura 4.8 mostra una rappresentazione di un raggio di ray casting colpire il volume. Il parametro *C(R)* indica il colore finale, mentre, *C(R,k)* e $\alpha(R,k)$ indicano rispettivamente, il colore e l'opacità durante la composizione. La componente *C* è calcolata secondo la formula di composizione *back-to-front*, analizzata nella sezione 4.6.3.

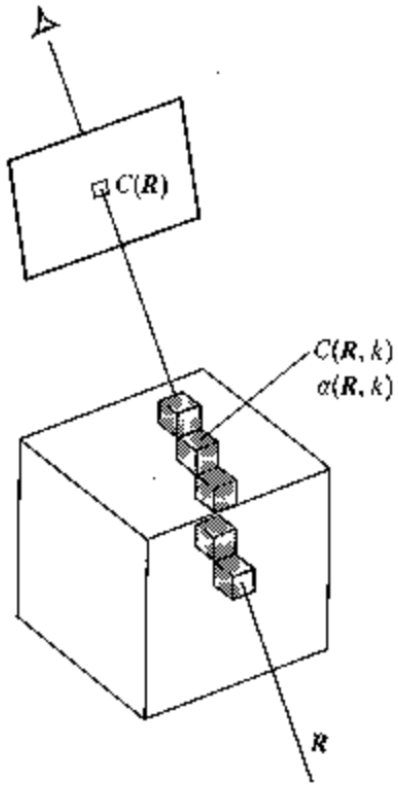


Figura 4.7 Un raggio di ray casting che interseca il volume.

Di seguito, si propone lo pseudocodice dell'algoritmo di ray casting [46]:

```
1. procedure RayCast(u) begin
2.     C(u) := 0;
3.     a(u) := 0;
4.     x1 := First(u);
5.     x2 := Last(u);
6.     u1 := Image(x1);
7.     u2 := Image(x2);
8.     // Loop through all samples falling within data
9.     for u := ul to u2 do begin
10.        x := Object(V);
11.        // Resample color and opacity and composite into ray
12.        C(U) := Sample(C,x);
13.        a(U) := Sample(a,x);
14.        C(u) := C(u) + C(U)(l - a(u));
15.        a(u) := a(u) + a(U)(l - a(u));
16.    end
17. end RayCast
```

Le funzioni *First* e *Last*, prendono come argomento l'indice del raggio e ritornano, rispettivamente, le coordinate del punto in cui il raggio entra ed esce dal volume. Le funzioni *Object* e *Image* hanno

lo scopo di convertire le coordinate, in *object-space* e in *image-space.* Infine, la funzione *Sample*, prende in input un array 3D di colori o di opacità e le coordinate di un punto (*object-space)*, e restituisce come output, un'approssimazione del valore di input tramite un'interpolazione trilineare degli otto voxel vicini [46].

Capitolo 5

Implementazione del progetto

In questo capitolo si parlerà dell'implementazione del progetto, fornendo una descrizione di tutti i componenti principali, delle funzionalità sviluppate, dei metodi di interazione con l'ambiente virtuale, e dell'interfaccia di utilizzo.

5.1 Descrizione dell'applicazione

L'applicazione prodotta è un prototipo sviluppato per l'HTC Vive, in cui sono implementate le caratteristiche necessarie per visualizzare e interagire con dati volumetrici. L'utilizzatore è immerso in un'ambiente virtuale estremamente realistico, con cui può interagire usando i controller di movimento. È infatti, possibile afferrare, manipolare e usare oggetti o strumenti virtuali di vario tipo. La simulazione ha luogo all'interno di una stanza di ospedale, nel quale è presente un paziente disteso su un letto. All'utente è data la possibilità di selezionare la scansione da visualizzare, contrassegnata da icone poste sul paziente in corrispondenza della parte del corpo interessata. Una volta selezionato quali dati mostrare, si avvierà una nuova scena, dove vi sono una serie di strumenti utilizzabili per manipolare i dati, in modo da ottenere una migliore comprensione per la diagnosi. Alcuni esempi di questi strumenti sono: un apparecchio in grado di nascondere a piacere una parte del volume, utile per osservare la composizione interna dei tessuti; un set di pennarelli 3D di colori diversi per marcare le parti d'interesse della scansione; una serie di pulsanti virtuali con svariate funzioni, tra cui la rotazione del volume su assi predefiniti. Ovviamente, l'utilizzatore potrà muoversi liberamente nel mondo fisico, e i medesimi movimenti saranno riprodotti nella simulazione grazie alla *room scale VR* (trattato nel capitolo 1). Tuttavia, si è ritenuto opportuno implementare un sistema di locomozione artificiale, atto a rendere più confortevole l'utilizzo, riducendo la fatica in caso di lunghe sessioni. Nelle sezioni successive, si entrerà nel dettaglio di tutte queste tematiche.

5.2 Architettura dell'applicazione

L'applicazione è costituita da un insieme di *GameObjects,* ognuno dei quali implementa una funzione specifica. In particolare, sono presenti 5 macro blocchi nella *Hierarchy* del progetto. Questi sono *VRTK*, *Volume*, *Models*, *Tools*, *Monitors*. All'interno di questi, tutti gli elementi sono raggruppati in maniera logica, per sfruttare il meccanismo della parentela, oppure per funzionalità simili; ad esempio, tutti gli strumenti per la visualizzazione sono raggruppati all'interno dell'oggetto *Tools*. Le funzionalità degli oggetti dipendono dai suoi componenti, i quali possono essere standard di Unity (es. *BoxCollider*), oppure, script sviluppati *ad hoc*.

Il diagramma della figura 5.1, mostra come sono strutturati gli oggetti all'interno della scena principale dell'applicazione, con i relativi componenti. L'oggetto *CameraRig*, composto da *Camera* e *Controller,* costituisce il blocco *VRTK* e permette all'utente di interfacciarsi con l'ambiente virtuale. L'oggetto *Volume* rappresenta i dati volumetrici da visualizzare, ad esso sono associate le *ViewCameras* che servono per mostrare le sezioni lungo i piani anatomici. Gli oggetti *ControlPanel*, *Slicers*, *DissectionTool* e *MarkingPen* sono gli strumenti per la manipolazione del volume, e hanno in comune il componente di interazione *VRTK_InteractableObject*. Infine, il blocco *Models* contiene i modelli 3D dell'ambiente, che rappresentati da oggetti statici non interagibili.

Durante lo sviluppo, si è cercato di mantenere un basso numero di dipendenze tra gli oggetti, in modo da non dover stravolgere l'intero programma in seguito a una singola modifica. Sempre dalla figura 5.1, è possibile osservare tre oggetti che presentano dipendenze: la *Camera* e il *ControlPanel* hanno bisogno di un riferimento all'oggetto *Volume*, il primo per effettuare il rendering, mentre il secondo per ruotare il volume mediante la pressione dei pulsanti; gli *Slicers* necessitano di una *ViewCamera* per selezionare la sezione da visualizzare sui *Monitors*, che avviene tramite le *RenderTexture.* Nelle sezioni successive, verranno approfondite le scelte di design e l'implementazione delle funzionalità del progetto.

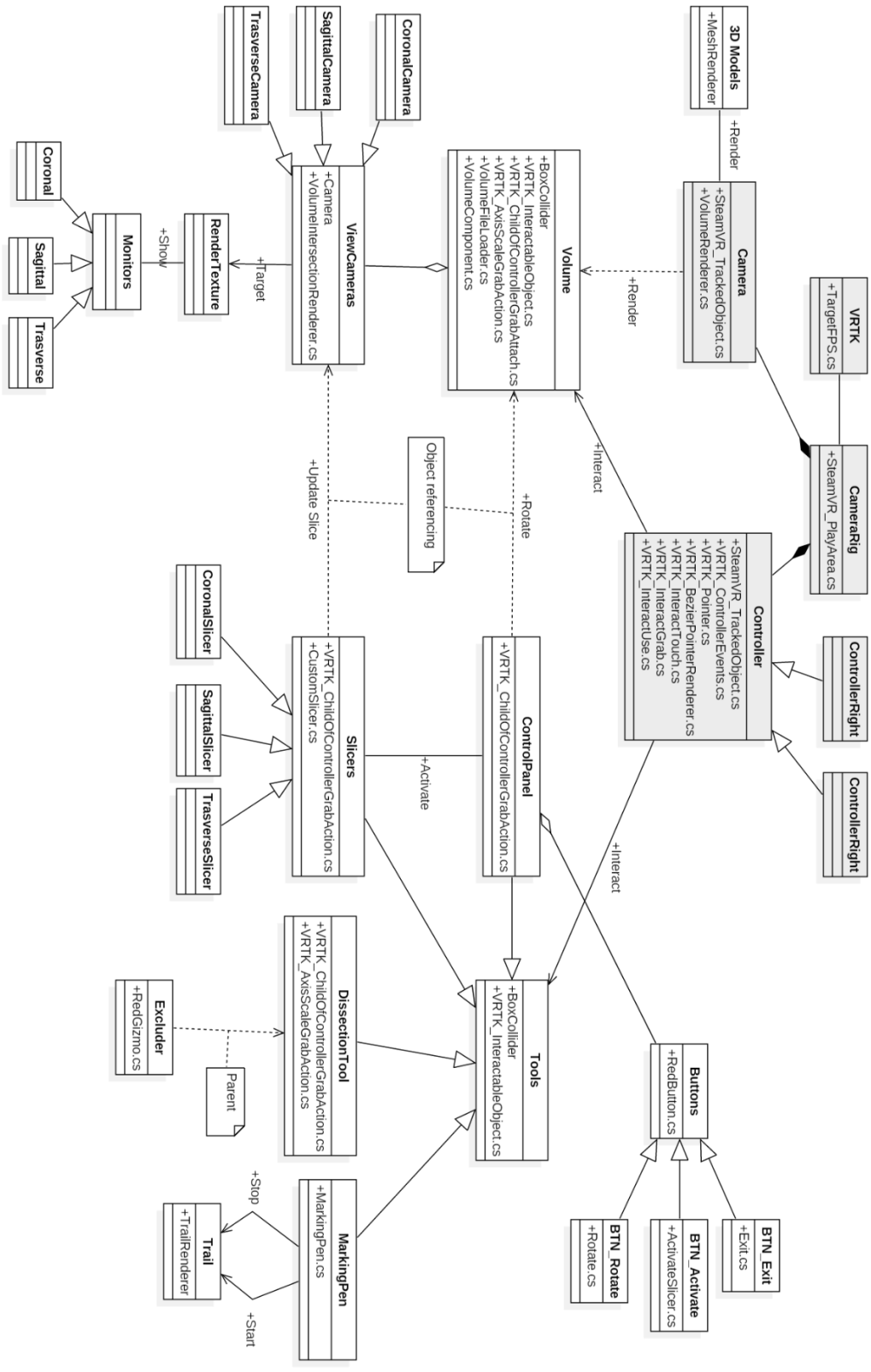


Figura 5.1 Diagramma degli oggetti e dei componenti dell'applicazione.

5.3 Le modalità di interazione

L'interazione è la caratteristica che contraddistingue la realtà virtuale dalla classica interfaccia desktop. Come già detto in precedenza, l'hardware dell'HTC Vive è composto dal visore e da due controller di movimento (figura 5.2), tutti tracciati nello spazio 3D grazie al sistema *Lighthouse*. Questo crea un nuovo paradigma di design delle applicazioni, perché i controller possono assumere qualsiasi funzionalità.

Il pattern seguito durante l'implementazione del progetto è quello di trattare i controller come l'estensione delle mani, assegnando quasi tutte le funzionalità ad oggetti con i quali si può interagire. In questo modo si semplifica l'apprendimento dell'applicazione, perché risulta molto più facile e intuitivo afferrare un oggetto che rappresenta una penna, piuttosto che selezionare la funzione di scrittura da un menù di un'interfaccia grafica sconosciuta.

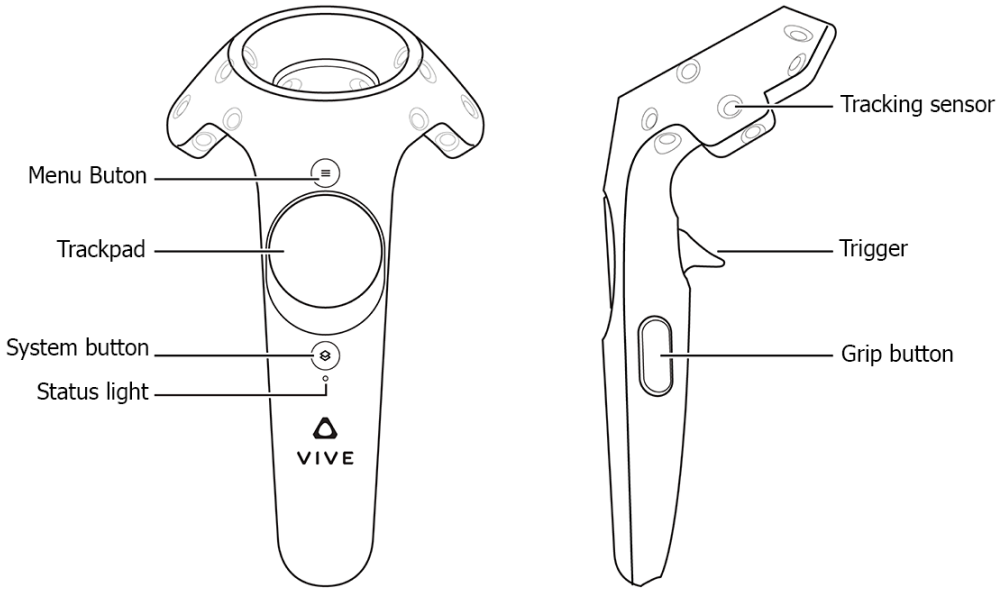


Figura 5.2 Il controller del Vive. I componenti usati per l'interazione sono: trackpad, grilletto (trigger) e il grip button. Il pulsante laterale (grip) simula la presa di un oggetto, mentre la pressione del grilletto indica l'utilizzo dell'oggetto in mano. Il trackpad è usato per attivare il sistema di locomozione artificiale.

5.3.1 Locomozione

Per locomozione si intende il sistema che permette il movimento all'interno dell'ambiente virtuale. Al momento, l'applicazione permette di muoversi in due modi: fisicamente o mediante la funzione

di teletrasporto. La prima, consiste nel muoversi realmente con addosso il visore, cosa del tutto sicura grazie al supporto offerto da SteamVR (sistema *Chaperone*), il quale proietta una griglia nel caso in cui ci si avvicina troppo ad una parete nel mondo reale.

Figura 5.3 Illustrazione del Sistema Chaperone. Avvicinandosi a una parete nel mondo fisico, per avvertire del potenziale pericolo, nella simulazione verrà mostrata una griglia ben visibile.

Per estendere il raggio di movimento oltre ai limiti dello spazio fisico, si è pensato di implementare un sistema di locomozione artificiale. Questo sistema, permette all'utente di teletrasportarsi nella posizione desiderata utilizzando un puntatore. Nel progetto, questa funzione è stata realizzata sfruttando la libreria VRTK, aggiungendo i componenti script *VTRK_pointer* e *VRTK_BezierPointerRender* ai game object di SteamVR che rappresentano i controller. Una volta attivata la funzione, attraverso la pressione del trackpad, dal controller si forma una curva di Bezier che finisce sul pavimento, questo indica il punto di arrivo del teletrasporto.

La motivazione dietro all'uso della curva di Bezier, è dovuta alla possibilità di designare un punto pur non avendo il contatto visivo; per esempio, se è presente un ostacolo tra il controller e la destinazione, non è possibile usare un puntatore di tipo laser. Invece, con un puntatore di Bezier, basterà muovere il controller in modo da aumentare l'angolo della curva per superare l'ostacolo.

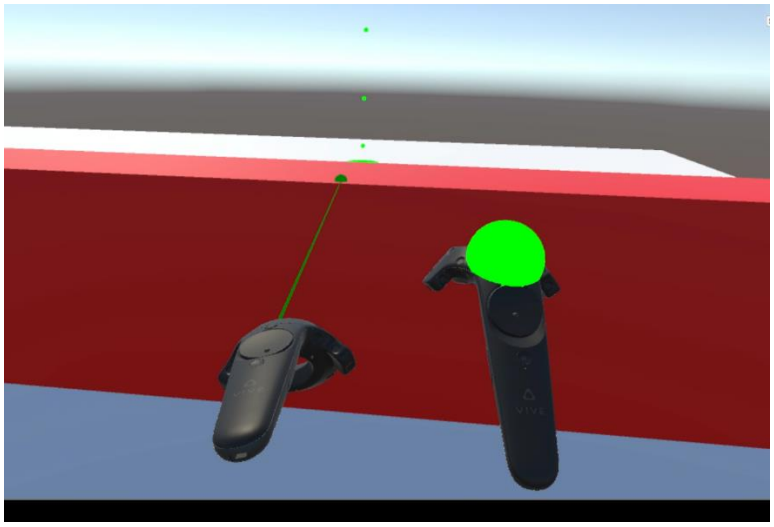


Figura 5.4 La funzione teletrasporto. Puntatore laser (sinistra) e Bezier (destra).

5.3.2 Interazione mediante il controller

Come discusso nel capitolo 2, l'accesso al controller è fornito dal plugin SteamVR. Per creare un riferimento al controller è necessario creare una variabile all'interno dello script:

```
1.  private SteamVR_TrackedObject trackedObj;
2.
3.  private SteamVR_Controller.Device Controller
4.  {
5.      get { return SteamVR_Controller.Input((int)trackedObj.index); }
6.  }
7.
8.  //Chiamata quando lo script viene caricato
9.  void Awake()
10. {
11.     trackedObj = GetComponent<SteamVR_TrackedObject>();
12. }
13.
14. //Chiamata ad ogni frame
15. void Update()
16. {
17.     //Pressione del grilletto
18.     if (Controller.GetHairTriggerDown()) {
19.        Debug.Log(gameObject.name + " Trigger Press");
20.     }
21.
22.     //Rilascio del grilletto
23.     if (Controller.GetHairTriggerUp()) {
24.         Debug.Log(gameObject.name + " Trigger Release");
25.     }
26. }
```

Una volta ottenuto il riferimento alla variabile *Controller*, sarà possibile utilizzarlo nella funzione *Update()* per implementare le funzionalità desiderate; per esempio, nello script precedente, alla pressione del grilletto viene stampato sulla console un messaggio di debug, ovviamente è possibile sviluppare azioni più complesse, combinando le giuste query.

Arrivati a questo punto, è evidente che lo sviluppo è molto dispendioso, in quanto bisogna programmare tutte le funzioni da zero, anche nel caso di quelle più basilari, come per esempio, la funzione di afferrare un oggetto. Fortunatamente, grazie alla libreria VRTK, non è necessario "reinventare la ruota". Infatti, con VRTK basterà estendere lo script *VRTK_InteractableObject* e inserire il codice della funzionalità nel metodo *StartUsing()*. Il codice seguente mostra come è stato realizzato lo script usato per caricare le scene:

```
1.  public class LoadScene : VRTK_InteractableObject
2.  {
3.      public int scene_id = 0;
4.
5.      public override void StartUsing(VRTK_InteractUse usingObject)
6.      {
7.          base.StartUsing(usingObject);
8.          load();
9.      }
10.
11.     private void load()
12.     {
13.         SceneManager.LoadScene(this.scene_id);
14.     }
15.
16. }
```

L'unica cosa richiesta è implementare la funzione *load()*, sarà poi compito di VRTK quello di chiamare *StartUsing()*, quando viene azionato l'oggetto a cui lo script è assegnato.

5.4 Le scene dell'applicazione

In Unity, una scena è un contenitore nella quale è possibile inserire qualsiasi tipologia di elemento. La si può immaginare come un insieme di GameObjects disposti in maniera ben precisa, e quindi permette di salvare e caricare le configurazioni per mostrare ambienti diversi. L'applicazione è composta da due scene: LobbyScene e InspectionScene.

5.4.1 LobbyScene

LobbyScene è la scena che viene caricata all'avvio dell'applicazione, è ambientata in una stanza di ospedale, dove all'interno è presente una rappresentazione del paziente disteso un letto. Questo setting funge come menù per scegliere la scansione da visualizzare, verranno mostrati sul corpo del paziente le parti disponibili per la visualizzazione. Per selezionare una scansione, è sufficiente posizionare il controller in corrispondenza dell'icona e premere il grilletto. A questo punto, l'applicazione caricherà l'InspectionScene per la visualizzazione.

Figura 5.5 Uno screenshot della LobbyScene visto dall'editor.

5.4.2 InspectionScene

L'InspectionScene è la scena principale dell'applicazione ed è composta dal volume da visualizzare e dagli strumenti di manipolazione. L'ambiente è lo stesso della LobbyScene, con la differenza che vengono tolti gli oggetti decorativi, come il letto e il divano, per dare spazio al volume e al carello di servizio, sul quale sono posti tutti gli strumenti per l'interazione. Questa scena, in sostanza, permette di visualizzare il volume e di manipolarlo allo scopo di effettuare una diagnosi.

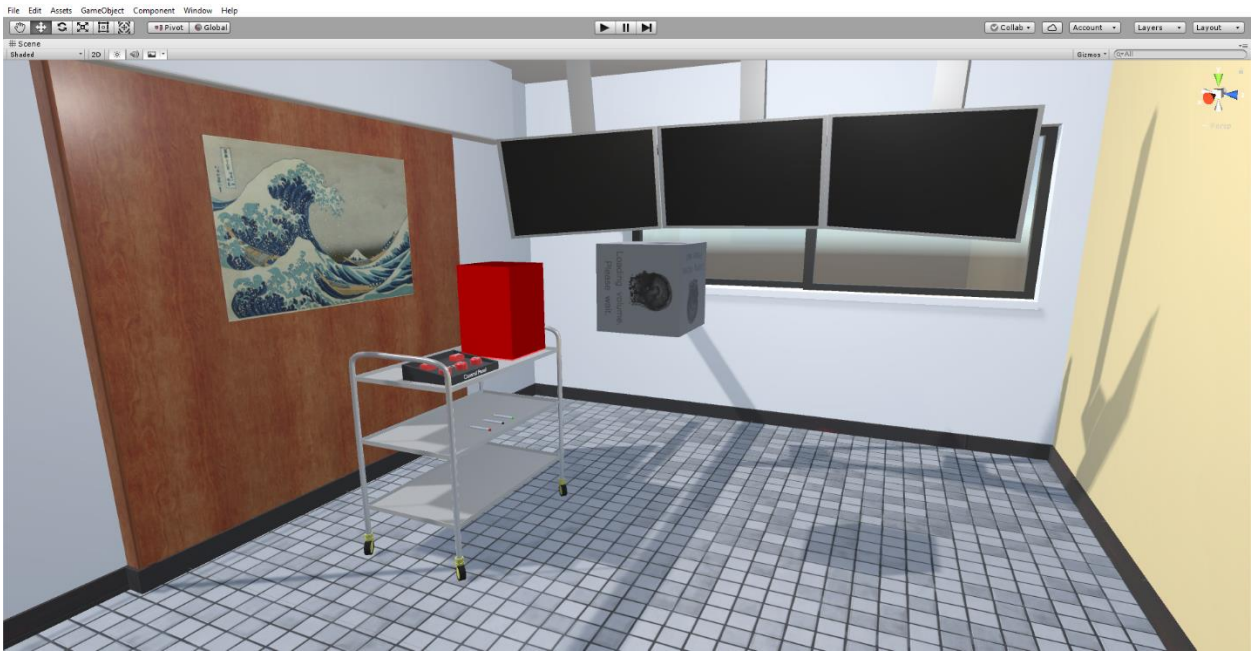


Figura 5.6 Uno screenshot dell'InspectionScene visto dall'editor.

5.5 I modelli 3D e i materiali

Tutti i modelli 3D, ad eccezione del paziente, sono stati creati usando il software Blender. Il processo di modellazione seguito è stato abbastanza lineare: prima si cerca un'immagine di riferimento del modello da realizzare, dopodiché si inserisce una primitiva (es. cubo) scalata in modo da avere le dimensioni corrette, infine, si modifica il poligono in modalità *Edit Mode* fino a raggiungere l'aspetto finale. Prima di importare il modello in Unity, è importante ricordare che quest'ultimo non supporta i materiali di Blender, ma solamente la mappatura UV delle texture.

Perciò i materiali devono essere per forza creati all'interno dell'editor di Unity, e le mappe UV editate con Blender. Uno dei modelli più complessi realizzati è stato il letto ospedaliero (figura 5.7), in quanto è composto da molteplici elementi, di cui alcuni, non hanno una forma regolare. In particolare, per ottenere l'effetto della coperta posta sul letto, si è dovuto ricorrere a una simulazione fisica di tipo *Cloth*, usando un oggetto *plane* suddiviso in segmenti sia in verticale che in orizzontale in modo da creare una griglia di vertici. A questo punto, la simulazione *Cloth* è in grado di muovere i vertici, simulando l'effetto di gravità.

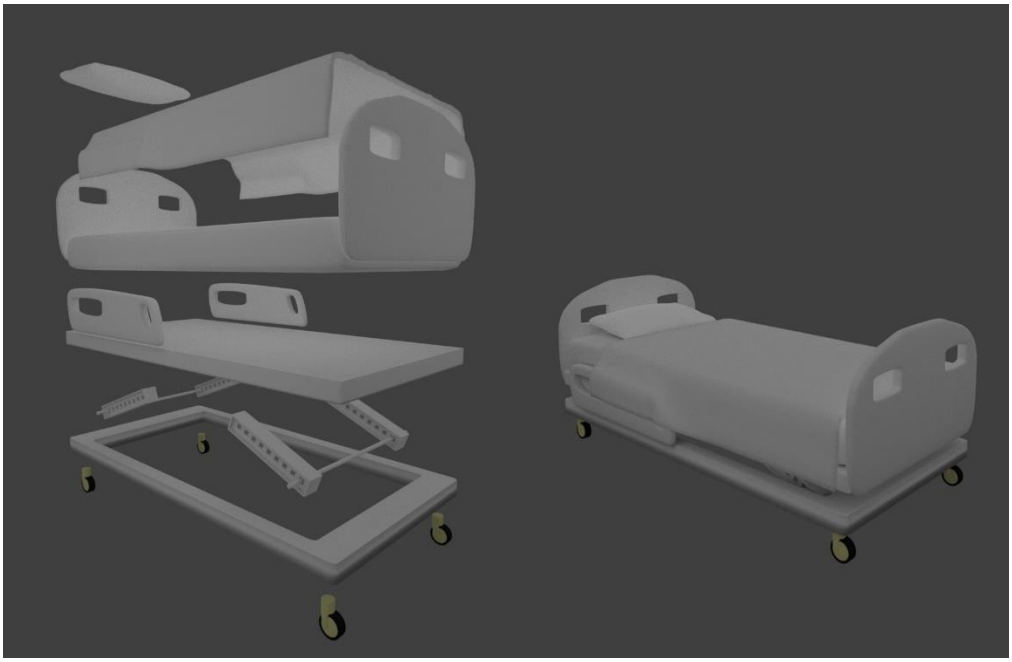


Figura 5.7 Il modello del letto ospedaliero e i suoi componenti, realizzato in Blender.

I materiali creati in Unity sono stati ottenuti dalla combinazione delle seguenti proprietà:

- **Rendering mode**: specifica la modalità di rendering, opaca o trasparente.
- **Albedo**: è il colore principale del materiale e può essere una texture colore.
- **Metallic**: valore che indica il grado di riflessione e può accettare una texture di riflessione.
- **Smoothness**: rappresenta la regolarità della superficie, ad esempio, uno specchio avrà un valore molto alto.
- **Normal**: prende in input una *normal map*, e serve per aggiungere dettagli superficiali senza aumentare il numero di poligoni.

5.6 La libreria VolumeViewer

VolumeViewer è una libreria per la visualizzazione di dati volumetrici in tempo reale. Essa è disponibile come package all'interno dell'asset store di Unity. VolumeViewer è in grado di visualizzare qualsiasi tipo di dati scalari o RGBA espressi come array tridimensionali, perciò le scansioni RM e TAC sono supportate [47]. La libreria utilizza l'algoritmo di volume Ray-Casting (discusso nel capitolo precedente) per creare una proiezione dei dati volumetrici. Successivamente, la proiezione viene inserita sull'immagine finale della camera. Alla presenza di multipli oggetti volumetrici, viene effettuata una proiezione ognuno di essi tenendo conto della distanza dalla camera, ovvero gli oggetti lontani verranno inseriti prima di quelli più vicini.
Le principali caratteristiche della libreria sono:

- Supporto al rendering di multipli volumi.
- Supporto al rendering con multiple camere.
- Possibilità di usare *transfer functions* monodimensionali e bidimensionali.
- Compatibilità con SteamVR e OpenVR.
- Possibilità di personalizzare vari parametri del volume, tra cui: il numero di *sample* da considerare lungo il raggio, contrasto, luminosità e opacità.
- Supporto di file: asset Texture3D, NIfTI e DICOM.

I componenti fondamentali per l'utilizzo di VolumeViewer sono *VolumeFileLoader*, *VolumeComponent* e *VolumeRenderer*.

5.6.1 VolumeFileLoader

Il componente *VolumeFileLoader* serve per caricare i dati volumetrici dai file. Si tratta di uno script che deve essere aggiunto al *GameObject* del volume, detto anche oggetto *proxy*. Per selezionare il file da caricare, bisogna cliccare sul tasto "*Choose*", presente nella finestra *inspector* di Unity, in corrispondenza della variabile *DataPath*.

Le variabili esposte (con visibilità public) più importanti sono:

- **loadOnStartup**, è un flag che, se impostato a *true*, carica il file nella funzione *Start(),* alternativamente, il file può essere da un altro script in un secondo momento;
- **dataPath**, memorizza il percorso del file;
- **loadAssetInstead**, è un riferimento ad un oggetto *VolumeComponentAsset*, il quale può essere usato al posto di dataPath.
- **loadingProgressChanged**, è un evento che manda lo stato di stato di caricamento (da 0 a 1) agli *event listeners.*

5.6.2 VolumeComponent

Lo script *VolumeComponent* include tutte le informazioni che determinano l'aspetto del volume. Anche questo componente deve essere aggiunto all'oggetto *proxy*. Inoltre, adotta un'architettura ad eventi: ad ogni cambiamento, vengono notificati tutti gli oggetti *listeners*.

Le variabili esposte più importanti sono:

- **resolution**, determina la risoluzione dell'immagine renderizzata. All'aumentare di questo valore (di default è 1) diminuisce la qualità, in quanto la risoluzione dell'immagine viene ottenuta dalla divisione delle dimensioni dello schermo per la *resolution*.
- **maxSample**, numero massimo di *samples* da effettuare lungo il raggio di Ray-Casting. Questo valore è direttamente proporzionale alla qualità finale dell'immagine.
- **voxelDimensions**, determina le dimensioni di un singolo voxel.
- **valueRangeMin**, rimappa i valori dei voxel secondo la formula [47]:
  $$(\min(voxel.rgb) - valueRangeMin) \div (valueRangeMax - valueRangeMin)$$
- **valueRangeMax**, rimappa i valori dei voxel secondo la formula [47]:
  $$(\max(voxel.rgb) - valueRangeMin) \div (valueRangeMax - valueRangeMin)$$
- **tfData**, indica la *transfer function* da applicare sui dati.
- **constrast**, determina il contrasto del volume.
- **brightness,** determina la luminosità del volume.
- **opacity,** determina l'opacità del volume.

5.6.3 VolumeRenderer

Il componente *VolumeRenderer* è responsabile per il rendering effettivo del volume. A differenza degli script precedenti, *VolumeRenderer* deve essere assegnato alla camera principale della scena. Esso presenta una sola variabile esposta: *volumeObjects*, il quale rappresenta la lista dei volumi da renderizzare. Il processo di rendering si basa sugli shaders; per ogni volume da renderizzare, all'oggetto *proxy* viene assegnato un materiale creato dallo shader *rayCastShader*, utilizzando i valori memorizzati nel *VolumeComponent.*

5.6.4 Integrazione di VolumeViewer in Unity

Gli step da seguire per iniziare ad usare la libreria VolumeViewer sono i seguenti:

1. Importare il package VolumeViewer dall'asset store di Unity.
2. Aggiungere, nell'editor, un nuovo layer chiamato "VolumeViewer".
3. Includere gli shaders nella *build.* L'opzione si trova in Edit->Project Settings->Graphics, dove all'array *AlwaysIncludedShaders* bisogna aggiungere tutti gli shaders presenti nella cartella Assets/VolumeViewer/Shaders/.
4. Creare l'oggetto proxy del volume, è sufficiente inserire un semplice cubo direttamente da Unity.
5. Aggiungere il componente *VolumeFileLoader* all'oggetto proxy e indicare il *dataPath* del percorso del file che si vuole usare.
6. Aggiungere lo script *VolumeComponent* all'oggetto proxy.
7. Aggiungere il componente *VolumeRenderer* alla camera principale della scena. Successivamente, aggiungere all'array *VolumeObjects* l'oggetto proxy, in modo da creare un riferimento.

A questo punto, è possibile avviare l'applicazione.

5.7 Gli oggetti interattivi

In questa parte, si discuterà l'implementazione dei principali oggetti con cui è possibile interagire all'interno dell'applicazione, che sono: il volume, lo strumento di dissezione (Dissection tool), i pennarelli 3D, lo slicer e i punti di controllo.

5.7.1 Il volume

Per la visualizzazione del volume è stata utilizzata la libreria VolumeViewer, che mette a disposizione una serie di script per il rendering. Il setup consiste, prima di tutto, nel creare un cubo che diventerà il contenitore del volume, assegnando lo script *VolumeLoader e VolumeComponent*, per caricare i dati volumetrici e designare l'oggetto come un volume. Successivamente, bisogna assegnare lo script VolumeRender alla camera della scena. Questo ha, effettivamente, il compito di visualizzare il volume, tramite il metodo di Ray-Casting.

5.7.2 Dissection tool

Lo strumento di dissezione serve per vedere l'interno del volume, in pratica, l'effetto che si ottiene è simile al risultato che si otterrebbe tagliando una parte del volume (figura 5.8).

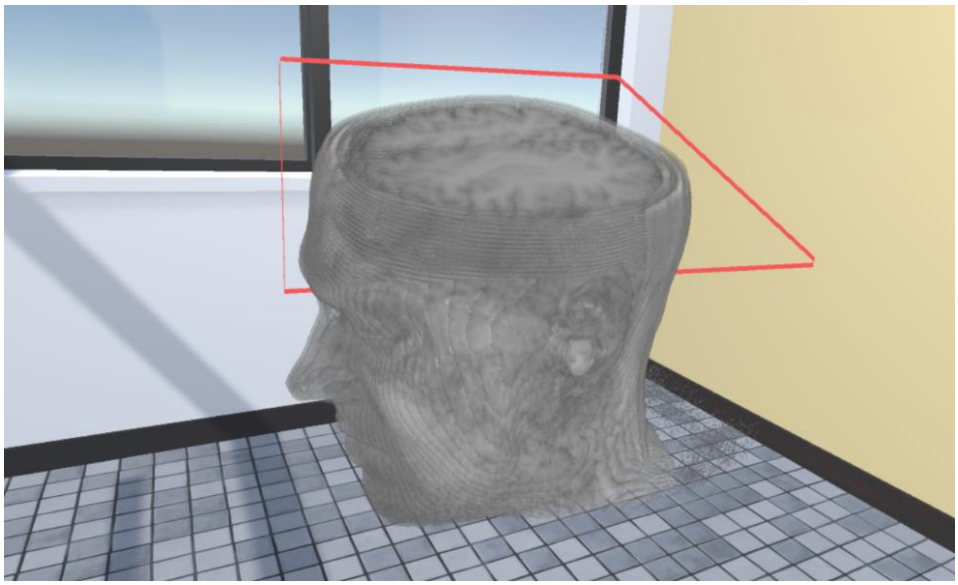


Figura 5.8 Lo strumento di dissezione in azione.

Lo strumento è caratterizzato da un oggetto a forma di parallelepipedo, che una volta posto sul volume, nasconde la parte intersecata. Lo script usato per la creazione di questo effetto è mostrato di seguito:

```
1.  public class RedGizmo : MonoBehaviour {
2.
3.  #if UNITY_EDITOR
4.      void OnDrawGizmos()
5.      {
6.          Gizmos.color = Color.red;
7.          Gizmos.DrawMesh(GetComponent<MeshFilter>().sharedMesh, transform.position, tran
    sform.rotation, transform.lossyScale);
8.      }
9.  #endif
10.
11. }
```

Come è possibile osservare, il codice di questo script crea una mesh trasparente al posto del parallelepipedo, e in quanto prevale sul modello del volume, anche parte di quest'ultima sarà resa trasparente. Inoltre, dato che non è possibile vedere questo oggetto, si è pensato di "imparentarlo" ad un cubo, scalato in modo opportuno e posizionato a un'estremità. Così facendo, ogni manipolazione del secondo oggetto verrà tramandata all'oggetto di dissezione, grazie al legame di parentela.

5.7.3 Il pennarello 3D

Il pennarello è uno strumento per fare annotazioni direttamente sul volume e in modo del tutto naturale. Infatti, una volta impugnato il pennarello, si potrà disegnare liberamente in tre dimensioni, come mostrato nella figura 5.9.

L'implementazione sfrutta il *LineRender* di Unity, che una volta agganciato a un oggetto, lascia una scia che segue il movimento dell'oggetto a cui è stato assegnato. L'idea, è quindi, quella di creare un oggetto scia (o trail) ad ogni pressione del grilletto, ma solamente se l'utente ha in mano il pennarello. La scia viene terminata al rilascio del grilletto, sfruttando anche qui, il meccanismo di parentela. Nella funzione *StopUsing* (override dalla classe *VRTK_InteractableObject*) viene tagliato il legame con l'oggetto scia, lasciandolo fermo in quel punto.

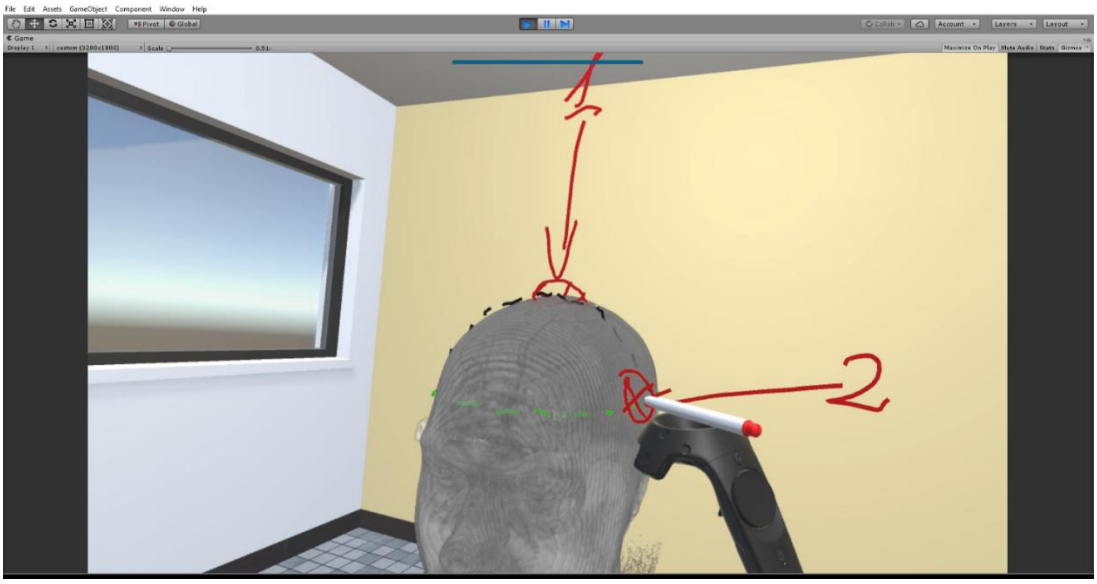


Figura 5.9 La funzione di disegno tramite lo strumento pennarello.

Lo script che permette di disegnare con il pennarello è *MarkingPen.cs*:

```
1.  public override void StartUsing(VRTK_InteractUse currentUsingObject)
2.  {
3.      base.StartUsing(currentUsingObject);
4.
5.      //Adds a new trail when user starts using the object
6.      addNewTrail();
7.  }
8.
9.  public override void StopUsing(VRTK_InteractUse previousUsingObject = null)
10. {
11.     base.StopUsing(previousUsingObject);
12.
13.     //Block the trail when the user stops using
14.     currentTrail.transform.parent = null;
15.     currentTrail = null;
16. }
17.
18. private void addNewTrail()
19. {
20.     if (currentTrail == null)
21.     {
22.         //Clone the trail and add to the pen as a child
23.         GameObject trailClone = Instantiate(trail, trail.transform.position, trail.tran
    sform.rotation) as GameObject;
24.         trailClone.SetActive(true);
25.
26.         currentTrail = trailClone;
27.         currentTrail.transform.parent = transform;
28.     }
29. }
```

5.7.4 Lo Slicer

Come i programmi di visualizzazione di dati medici tradizionali, l'applicazione permette la visione delle singole fette lungo i piani anatomici (figura 5.10), che sono costituiti dal piano sagittale, traversale, e da quello coronale (o frontale).

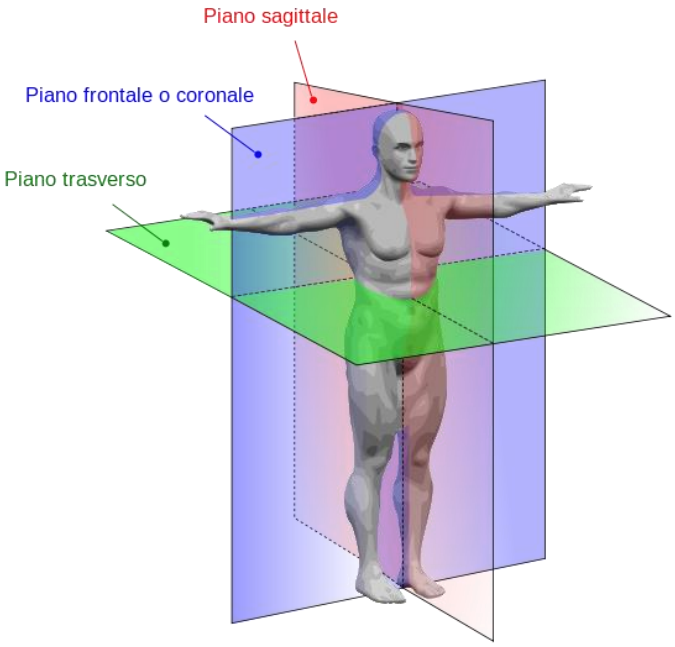


Figura 5.10 Piani anatomici.

Sono quindi presenti, nella scena, tre monitor virtuali (figura 5.11) che mostrano in tempo reale le fette selezionate dall'intersezione dei piani con il volume. Naturalmente, muovendo i vari piani con il controller, la vista dei monitor verrà aggiornata di conseguenza.

Al volume sono associate tre camere aggiuntive, una per ogni asse, e l'output di ognuna di esse è salvato su una *RenderTexture*. Si tratta di una texture speciale che viene aggiornata a runtime. Assegnando questa texture all'oggetto monitor, otteniamo uno schermo che mostra quello che vede la camera. Infine, sfruttando il *frustum* o volume di vista della camera, in particolare il piano di clipping frontale, è possibile visualizzare il volume a partire da una fetta. Quindi, spostando la camera o il *frustum*, otteniamo le sezioni lungo l'asse.

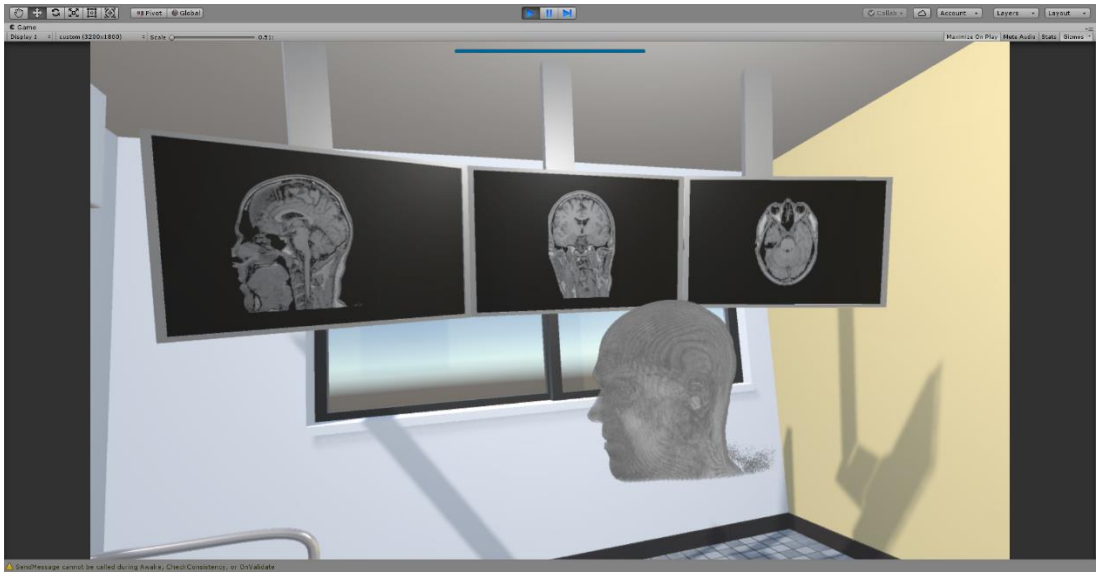


Figura 5.11 Lo strumento slicer in azione.

Le camere sono impostate in modalità ortografica, in quanto il volume di vista a forma di parallelepipedo risulta più idoneo rispetto alla quella piramidale della proiezione prospettica. Inoltre, per evitare la sovrapposizione di altri oggetti, le camere renderizzano solamente il layer del volume.

5.8 Applicazione in esecuzione

Una volta indossato il visore e avviata l'applicazione, l'utente si trova immerso in un ambiente virtuale che simula la stanza di un ospedale.

Figura 5.12 L'ambiente mostrato all'avvio

A questo punto l'utente può avvicinarsi al paziente disteso sul letto e selezionare il volume da mostrare.

Figura 5.13 Selezione del volume da visualizzare

Una volta selezionato il volume da visualizzare, l'utente è trasportato nella scena di ispezione, dove è possibile manipolare la scansione utilizzando gli strumenti descritti precedentemente.

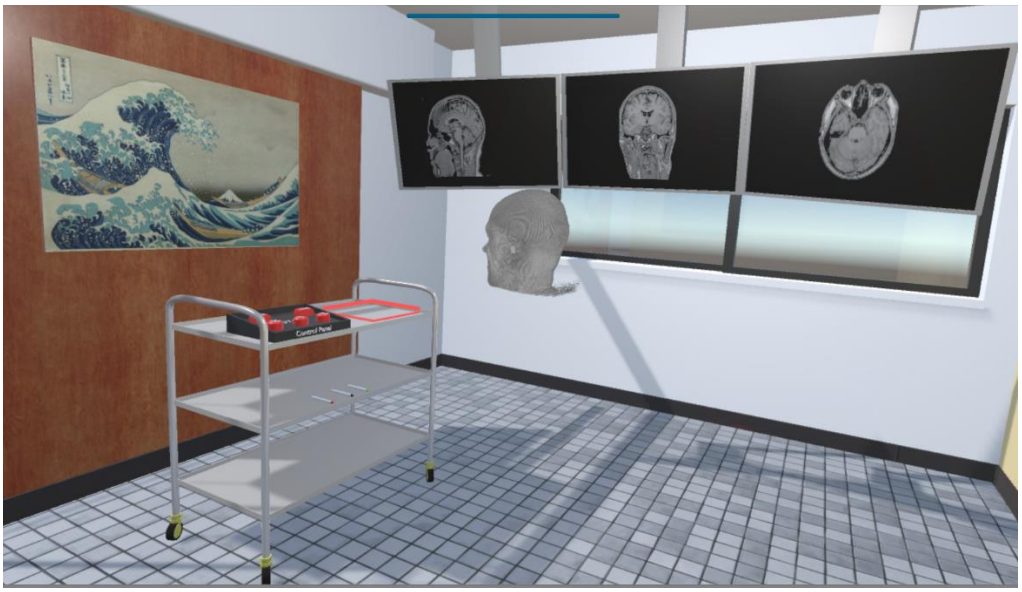


Figura 5.14 L'ambiente di visualizzazione del volume

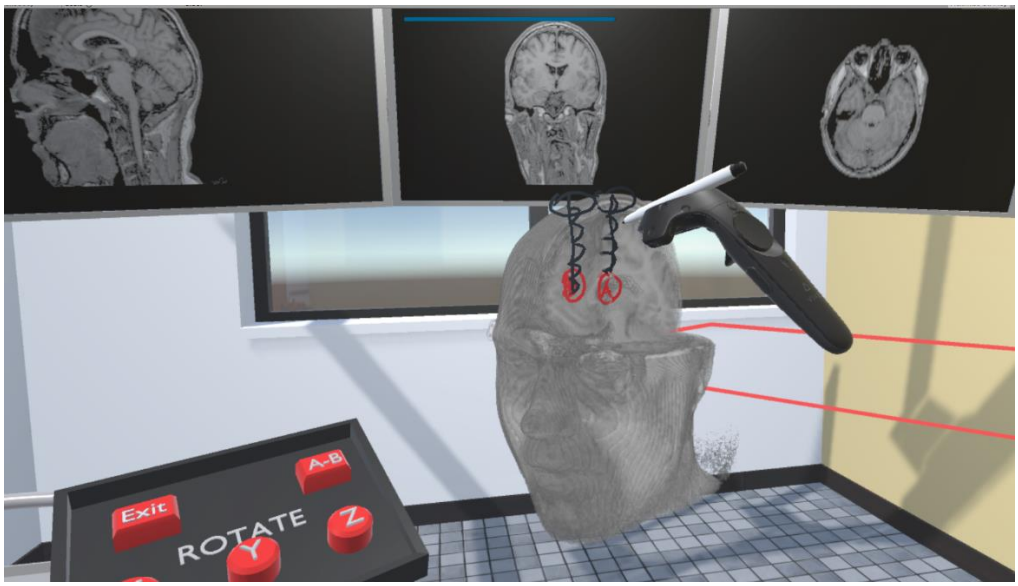


Figura 5.15 Manipolazione del volume combinando gli strumenti a disposizione.

Conclusioni e sviluppi futuri

Lo scopo del lavoro presentato è quello di valutare i moderni sistemi per la realtà virtuale, nel contesto della diagnostica per immagini. Si ritiene che tecnologie di questo tipo possano essere estremamente utili per assistere i medici nella diagnosi, in quanto, grazie all'immersione in un ambiente virtuale, sono in grado di mostrare più informazioni simultaneamente.

Il progetto sviluppato è un'applicazione di realtà virtuale, che unisce le tipiche caratteristiche di interazione della VR con la visualizzazione volumetrica di dati tomografici. In particolare, è stato creato una simulazione, dove è possibile interagire con il volume. Le azioni implementate sono: movimento, rotazione, *scaling*, dissezione, analisi delle fette lungo i piani anatomici, e annotazioni con il pennarello 3D.

Dopo aver esposto le funzioni implementate, si può concludere che i requisiti iniziali sono stati pienamente soddisfatti.

Tuttavia, l'applicazione ancora non può definirsi completa, oltre all'ottimizzazione del codice, uno dei miglioramenti potrebbe essere l'implementazione di un sistema di salvataggio delle sessioni e delle modifiche apportate al volume. Un'altra miglioria potrebbe riguardare l'aumento della definizione del volume, e conseguentemente, il supporto a file di dimensioni maggiori. Sarebbe inoltre implementabile l'opzione di dare all'utente la possibilità di definire le *trasfer functions*, in modo da evidenziare zone particolari del volume. Infine, un ulteriore sviluppo potrebbe riguardare l'introduzione di una modalità multiplayer, dove più medici cooperano per effettuare, in tempo reale, la diagnosi dello stesso volume.

Bibliografia e sitografia

[1] Andrea Carobene. Realtà virtuale. In Enciclopedia online Treccani.

[2] Sega VR. https://segaretro.org/Sega_VR

[3] Virtual Boy. http://nintendo.wikia.com/wiki/Virtual_Boy

[4] Yuval Boger. https://www.roadtovr.com/understanding-pixel-density-retinal-resolution-and-why-its-important-for-vr-and-ar-headsets/

[5] Brooklyn Waters. http://mtechgames.com/downloads/PhysicsandFramerateBeatingmotionsicknessinVR.pdf

[6] Oculus best practices. Pag.4. https://static.oculus.com/documentation/pdfs/intro-vr/latest/bp.pdf

[7] Alex Vlachos, Valve. Advanced VR Rendering. GDC 2015.

[8] HTC Vive. https://www.vive.com/us/product/vive-virtual-reality-system/

[9] Oliver Kreylos. UC Davis. http://doc-ok.org/?p=1478

[10] Lora Kolodny. TechCrunch. https://techcrunch.com/2017/03/26/nasa-is-using-a-mixed-reality-space-station-to-train-astronauts/

[11] Unity Features. https://unity3d.com/unity

[12] Unity Platforms. https://unity3d.com/unity/features/multiplatform

[13] Unity Transform component. https://docs.unity3d.com/Manual/Transforms.html

[14] Script Lifecycle Flowchart. https://docs.unity3d.com/Manual/ExecutionOrder.html

[15] Precalculated Realtime GI. https://docs.unity3d.com/Manual/GIIntro.html

[16] OpenVR API documentation. Valve software. https://github.com/ValveSoftware/openvr/wiki/API-Documentation

[17] VRTK Documentation. https://vrtoolkit.readme.io/docs/summary

[18] Jerrold T. Bushberg. The Essential Physics of Medical Imaging

[19] Stefano Pacifici. Lo standard di qualità nella mammografia di screening.

[20] Mark Hammer. MRI Physics. http://xrayphysics.com/spatial.html

[21] Humanitas Reasearch Hospital. http://www.humanitas.it/visite-esami/ecografia

[22] *RadiologyInfo.org*. General Nuclear Medicine. https://www.radiologyinfo.org/en/info.cfm?pg=gennuclear

[23] C. T. Walker and R. O. Pohl. Photon Scattering By Point Defects, 1963.

[24] Andrew Zimmerman Jones. The Compton effect in Physics. https://www.thoughtco.com/the-compton-effect-in-physics-2699350

[25] Khan Academy. Photoelectric effect. https://www.khanacademy.org/science/physics/quantum-physics/photons/a/photoelectric-effect

[26] Dr. Eng. Sarah Hagi. CT-Generations, RAD309. https://www.kau.edu.sa/Files/0008512/Files/19500_2nd_presentation_final.pdf

[27] Liu Y. (2014) Magnetic Resonance Imaging. Current Laboratory Methods in Neuroscience Research. Springer Protocols Handbooks. Springer, New York, NY

[28] Sergio Pissanetzky. Minimum energy MRI gradient coils of general geometry. Texas Accelerator Center.

[29] Michele Larobina, Loredana Murino. Medical Image File Formats. CNR Napoli. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3948928/pdf/10278_2013_Article_9657.pdf

[30] Anderson Winkler. National Institutes of Health. https://brainder.org/2012/09/23/the-nifti-file-format/

[31] Philippe G. Lacroute. Fast volume rendering using a shear-warp factorization of the viewing transformation. Stanford University 1995.

[32] Teorema del campionamento. https://it.wikipedia.org/wiki/Teorema_del_campionamento_di_Nyquist-Shannon

[33] Tomas Akenine-Moller, Tomas Moller, and Eric Haines, Real-time rendering, A. K. Peters, Ltd., 2002.

[34] Tom Roelandts. How to create a simple low-pass filter. https://tomroelandts.com/articles/how-to-create-a-simple-low-pass-filter

[35] Marco Bosma. Iso-surface rendering. University of Twente. https://ris.utwente.nl/ws/portalfiles/portal/6040217

[36] Paul S. Calhoun, BFA, Brian S. Kuszyk, MD, David G. Heath, PhD, Jennifer C. Carley, BS, and Elliot K. Fishman, MD. InfoRad 1997. http://pubs.rsna.org/doi/full/10.1148/radiographics.19.3.g99ma14745

[37] Çelebi Engineering. https://www.byclb.com/TR/Tutorials/volume_rendering/ch1_1.htm

[38] Timothy J. Cullip and Ulrich Neumann. Accelerating volume reconstruction with 3d texture hardware. Technical Report: Univ. of North Carolina at Chapel Hill, 1994.

[39] M. HADWIGER, J. M. KNISS, K. ENGEL, and C. REZK-SALAMA, High-Quality Volume Graphics on Consumer PC Hardware, SIGGRAPH, 2002.

[40] J. F Blinn, Light refelction functions for simulation of clouds and dusty surfaces., Computer Graphics 16(3) (1982).

[41] Marcus Jonsson. Umea University. http://www8.cs.umu.se/education/examina/Rapporter/MarcusJonsson.pdf

[42] Peter Lücke. Volume Rendering Techniques for Medical Imaging. http://campar.in.tum.de/twiki/pub/Students/DaLuecke/Diplomarbeit.pdf

[43] Tor Oyvind Fluor. Multidimensional Transfer Function in Volume Rendering of Medical Datasets. University of Oslo. https://www.duo.uio.no/bitstream/handle/10852/9416/Fluor.pdf

[44] Barthold Lichtenbelt, Randy Crane, and Shaz Naqvi. Introduction to Volume Rendering. Prentice-Hall, 1998.

[45] Nelson Max. Optical Models for Direct Volume Rendering. IEEE Transaction on Visualization and computer graphics, VOL. 1, NO.2, JUNE 1995.

[46] Marc Levoy. Efficient ray tracing of volume data, 1988. Univerisity of North Carolina.

[47] LISCINTEC. VolumeViewer Documentation.